(12)特許協力条約に基づいて公開された国際出願

(19) 世界知的所有権機関
国際事務局

(43) 国際公開日
2004年5月6日 (06.05.2004)

PCT

(10) 国際公開番号
WO 2004/038620 A1

(51) 国際特許分類[7]: G06F 17/50

(21) 国際出願番号: PCT/JP2003/012840

(22) 国際出願日: 2003年10月7日 (07.10.2003)

(25) 国際出願の言語: 日本語

(26) 国際公開の言語: 日本語

(30) 優先権データ:
特願2002-313201
2002年10月28日 (28.10.2002) JP

(71) 出願人 (米国を除く全ての指定国について): 株式会社ルネサステクノロジ (RENESAS TECHNOLOGY CORP.) [JP/JP]; 〒100-6334 東京都 千代田区 丸の内二丁目4番1号 Tokyo (JP).

(72) 発明者; および

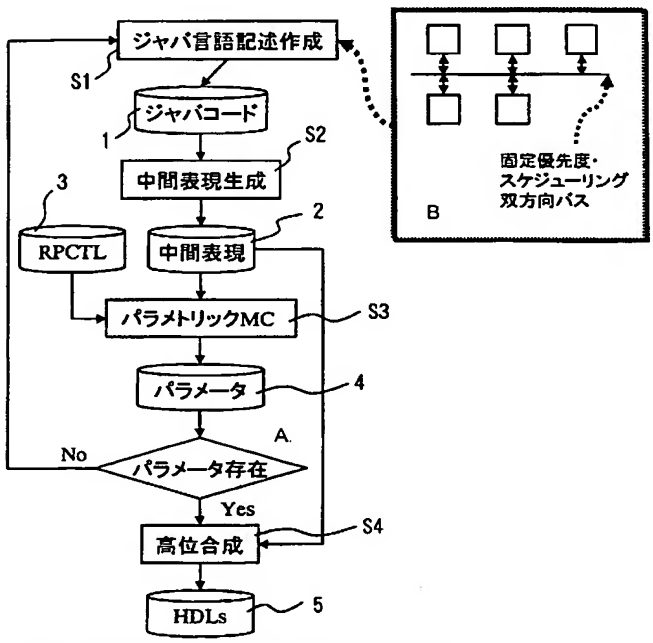
(75) 発明者/出願人 (米国についてのみ): 谷本 匡亮 (TANIMOTO,Tadaaki) [JP/JP]; 〒100-6334 東京都 千代田区 丸の内二丁目4番1号 株式会社ルネサステクノロジ内 Tokyo (JP). 鎌田 丈良夫 (KAMADA,Masurao) [JP/JP]; 〒100-6334 東京都 千代田区 丸の内二丁目4番1号 株式会社ルネサステクノロジ内 Tokyo (JP).

(74) 代理人: 玉村 静世 (TAMAMURA,Shizuyo); 〒101-0052 東京都 千代田区 神田小川町2丁目10番地 新山城ビル42号 Tokyo (JP).

[続葉有]

(54) Title: SYSTEM DEVELOPMENT METHOD AND DATA PROCESSING SYSTEM

(54) 発明の名称: システム開発方法及びデータ処理システム



S1...JAVA LANGUAGE DESCRIPTION GENERATION
1...JAVA CODE
S2...INTERMEDIATE EXPRESSION GENERATION
2...INTERMEDIATE EXPRESSION
S3...PARAMETRIC MC
4...PARAMETER
A...PARAMETER EXISTS?
S4...HIGH SYNTHESIS
B...FIXED PRIORITY/SCHEDULING BIDIRECTIONAL BUS

(57) Abstract: A program description (1) defining a plurality of devices by using a program language capable of describing a parallel operation is input. The program description which has been input is converted into an intermediate expression (S2). A parameter satisfying a real time restriction is generated for the intermediate expression (S3). According to the parameter generated, a circuit description by the hardware description language is synthesized (S4). The intermediate expression is a concurrent control flow flag, a time automaton having a concurrent parameter, or the like. In the aforementioned parameter generation, parametric model checking is performed. The program description is performed by defining a device by using the run method and defining a device clock synchronization by using the barrier synchronization. Thus, it is possible to design a bus system satisfying the real time restriction.

(57) 要約: 並列動作の記述が可能なプログラム言語を用いて複数のデバイスを定義したプログラム記述（1）を入力し、入力したプログラム記述を中間表現に変換し（S2）、中間表現に対し、実時間制約を満足するパラメータを生成し（S3）、生成したパラメータに基づいてハードウェア記述言語による回路記述を合成する（S4）。中間表現は、コンカレントなコントロールフローフラグ、コンカレントなパラメータ付き時間オートマトン等である。前記パラメータ生成に、パラメトリック・モデルチェッキングを行う。プログラム記述は、ｒｕｎメソッドを用いてデバイスの定義を行い、バリア同期を用いてデバイスのクロック同期を定義する。これにより、実時間制約を満たすバス・システムを設計することができる。

WO 2004/038620 A1

(81) 指定国 (国内): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) 指定国 (広域): ARIPO 特許 (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), ユーラシア特許 (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), ヨーロッパ特許 (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI 特許 (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

添付公開書類:
— 国際調査報告書

２文字コード及び他の略語については、定期発行される各*PCT*ガゼットの巻頭に掲載されている「コードと略語のガイダンスノート」を参照。

明　細　書

システム開発方法及びデータ処理システム

技術分野

本発明は、並列動作を記述できる言語からディジタル回路を開発する方法、更には並列動作を記述できる言語からディジタル回路のハードウェア合成を行うデータ処理システムに関する。

背景技術

近年、モバイル・コンピューティング環境を実現する上で、システムＬＳＩの果たす役割はその重要性を増している。また、モバイル・コンピューティングでは実時間制約を如何に満たすかが、しばしば問題となる。更に、システムＬＳＩを実装する上で要求性能を満たすよう設計する場合、バス・システムの設計が重要となる。然るに、バス・システムを実時間制約を満たすよう効率良く設計する為の設計手法がシステム・シミュレーションによる方法以外提案されていないのが現状である。システムシミュレーションについて記載された文献の例として下記の特許文献がある。

特許文献１：特開２００２－２７９３３３号公報

特許文献２：特開２０００－０３５８９８号公報

特許文献３：特開平０７－０８４８３２号公報

発明の開示

本発明の目的は、Ｊａｖａ（登録商標）等の並列動作を記述可能なプログラム言語を用いてバス・システム等のハードウェア設計の工数を低

減することにある。

本発明の目的は、並列動作を記述可能なプログラム言語とパラメトリック・モデルチェッキングを用いた、実時間制約を満たすバス・システムの新規設計手法を提供することにある。

本発明の別の目的は、モデルチェッキング技術とハードウェア合成技術の融合による新たな設計手法を提供することにある。

本発明のさらに別の目的は、実時間制約を有するバス・システムに対する並列動作記述可能なプログラム言語によるモデル化とパラメトリック・モデルチェックングによる検証、更にはハードウェア合成を実現することにある。

本発明の前記並びにその他の目的と新規な特徴は本明細書の記述及び添付図面から明らかになるであろう。

本願において開示される発明のうち代表的なものの概要を簡単に説明すれば下記の通りである。

すなわち、並列動作の記述が可能なプログラム言語を用いて複数のデバイスを定義したプログラム記述を入力し、入力したプログラム記述を中間表現に変換し、この中間表現に対し、実時間制約を満足するパラメータを生成し、生成したパラメータに基づいてハードウェア記述言語による回路記述を合成する。

上記より、Ｊａｖａ（登録商標）等の並列動作を記述可能なプログラム言語によりバスシステム等に対するモデル化を行って、実時間制約を満たすシステムの設計を行うことができる。これにより、ハードウェア設計の工数低減が可能になる。

本発明の一つの形態として、前記中間表現は、コンカレントなコントロールフローフラグ、コンカレントなパラメータ付き時間オートマトン、又はパラメータ付き時間オートマトンである。

本発明の一つの形態として、前記パラメータ生成に、パラメトリック・モデルチェッキングを行う。モデルチェッキング技術とハードウェア合成技術の融合による新たな設計手法を提供することができる。

本発明の一つの形態として、前記実時間制約はＲＰＣＴＬで与えられる。

本発明の一つの形態として、前記プログラム記述はｒｕｎメソッドを用いてデバイスの定義を行い、バリア同期を用いてデバイスのクロック同期を定義する。

図面の簡単な説明

第１図は本発明に係る設計方法の全体を例示するフローチャートである。

第２図は単一バス・システムのジャバ言語によるモデル化のデザインパターンを例示する説明図である。

第３図はモデル化におけるクロック同期メカニズムとしてバリア同期によるクロック同期メソッドを用いた例を示す説明図である。

第４図はレジスタへの値書込みを実現するモデル化の記述を例示する説明図である。

第５図はバス権獲得を管理するメソッドを例示する説明図である。

第６図は第５図の各メソッドの呼び出し関係を表すコールグラフである。

第７図はバス権獲得メカニズムのジャバ言語記述を例示する説明図である。

第８図は第７図の続きを示すジャバ言語記述の説明図である。

第９図はバス権解放を管理するメソッドの説明図である。

第１０図は第９図の各メソッドの呼び出し関係を表すコールグラフ

である。

第１１図はバス権解放メカニズムのジャバ言語記述を例示する説明図である。

第１２図は sync_read メソッドのジャバ言語コードの説明図である。

第１３図は run()での sync_read メソッド記述例を示す説明図である。

第１４図は sync_burst_read のジャバ言語コードを示す説明図である。

第１５図は endBurstAccess のジャバ言語コードを示す説明図である。

第１６図は freeBurstBusLock のジャバ言語コードを示す説明図である。

第１７図は run()でのバーストリードの記述例を示を説明図である。

第１８図は sync_write のジャバ言語コードを示す説明図である。

第１９図は run()での sync_write メソッド記述例を示す説明図である。

第２０図は sync_burst_write のジャバ言語コードを示す説明図である。

第２１図は run()でのバーストライトの記述例を示す説明図である。

第２２図はジャバ言語記述による実装例の概略仕様を示す説明図である。

第２３図はコマンドインタフェースに関する run() method の実装例の一部を示す説明図である。

第２４図は第２３図の実装例の続きを示す説明図である。

第２５図は第２４図の実装例の続きを示す説明図である。

第２６図は第２５図の実装例の続きを示す説明図である。

第２７図はユニファイドメモリに関する run() method の実装例を示す説明図である。

第２８図はグラフィック描画ユニットに関する run() method の実装例を示す説明図である。

第２９図は第２８図の実装例の続きを示す説明図である。

第３０図は第２９図の実装例の続きを示す説明図である。

第３１図は第３０図の実装例の続きを示す説明図である。

第３２図は第３１図の実装例の続きを示す説明図である。

第３３図は表示ユニットに関する run() method の実装例を示す説明図である。

第３４図は第３３図の実装例の続きを示す説明図である。

第３５図は中間表現への変換工程の詳細を全体的に例示する説明図である。

第３６図は C-CFG の形式を例示する説明図である。

第３７図は synchronized の扱い（ハード合成用）について示す説明図である。

第３８図は synchronized の扱い（パラメトリック・モデルチェッキング用）について示す説明図である。

第３９図は Command Interface の場合のＣＦＧを示す説明図である。

第４０図は Command Interface に関しハード合成用のＣＦＧを例示する説明図である。

第４１図はパラメトリック・モデルチェッキング用のＣＦＧを例示する説明図である。

第４２図はグラフィック描画ユニット (Graphics Rendering Unit) に関するＣＦＧを例示する説明図である。

第４３図は第４２図のＣＦＧの続を示す説明図である。

第４４図はパラメトリック・モデルチェッキング用のＣＦＧを例示する説明図である。

第４５図は第４５図のＣＦＧの続を示す説明図である。

第４６図は前記表示ユニット（Display Unit）に関するＣＦＧを例示する説明図である。

第４７図はパラメトリック・モデルチェッキング用のＣＦＧを例示する説明図である。

第４８図はコマンドインタフェース（Command Interface）のＣＦＧ、グラフィック描画ユニット（Graphics Rendering Unit）のＣＦＧ、及び前記表示ユニット（Display Unit）のＣＦＧにおける夫々の開始ノードの結合状態を例示する説明図である。

第４９図は固定優先度スケジューラを例示する説明図である。

第５０図はコマンドインタフェース（Command Interface）のＣＦＧに対する抽象化の様子を順を追って示す最初の説明図である。

第５１図は第５０図の続を示す説明図である。

第５２図は第５１図の続を示す説明図である。

第５３図は第５２図の続を示す説明図である。

第５４図は第５３図の続を示す説明図である。

第５５図は第５４図の続を示す説明図である。

第５６図は第５５図の続を示す説明図である。

第５７図はグラフィック描画ユニット（Graphics Rendering Unit）のＣＦＧに対する抽象化処理の結果を例示する説明図である。

第５８図は表示ユニット（Display Unit）のＣＦＧに対する抽象化処理の結果を例示する説明図である。

第５９図はコマンドインタフェース（Command Interface）のＣＦＧに対するＴＮＦＡへの変換の様子を順を追って示す説明図である。

第６０図は第５９図の続を示す説明図である。

第６１図は第６０図の続を示す説明図である。

第６２図はグラフィック描画ユニット（Graphics Rendering Unit）のＣＦＧに対するＴＮＦＡへの変換の様子を順を追って示す説明図である。

第６３図は第６２図の続を示す説明図である。

第６４図は第６３図の続を示す説明図である。

第６５図は第６４図の続を示す説明図である。

第６６図は表示ユニット（Display Unit）のＣＦＧに対するＴＮＦＡへの変換の様子を順を追って例示する説明図である。

第６７図は第６６図の続を示す説明図である。

第６８図は第６７図の続を示す説明図である。

第６９図は第６１図、第６４図、第６８図で求めたＴＮＦＡの積ＴＮＦＡの構成例を示す説明図である。

第７０図はパラメータに上限値を与えた時に、積ＴＮＦＡを構成する段階で制約を満たさない遷移枝の削除過程を示す説明図である。

第７１図は第７０図の続を示す説明図である。

第７２図は第７１図の続を示す説明図である。

第７３図は第７２図の続を示す説明図である。

第７４図は第７３図の続を示す説明図である。

第７５図は第７４図の続を示す説明図である。

第７６図は第７５図の続を示す説明図である。

第７７図はC-TNFA2TNFAでAbstraction of TNFAを適時コールした場合の処理の経過の一部を示す説明図である。

第７８図は第７７図の続を示す説明図である。

第７９図は第７８図の続を示す説明図である。

第８０図は得られたパラメータ条件に対して目的関数を各パラメタ値の合計として、これを最小化するという線形計画問題を解いた結果を

例示する説明図である。

第８１図は第８０図の結果に対し、kb1=2,kb2=1,kb3=1 の制約を追加して得た解を例示する説明図である。

第８２図は第８１図の結果に対し更に kr1 以外のパラメタ変数を１以上という制約を追加して得た解を例示する説明図である。

第８３図は BasicBlock への実行サイクルの割り当てを例示する説明図である。

第８４図は固定優先度スケジューラを例示する説明図である。

第８５図は第８４図に示す固定優先度スケジューラの一部を用いて変形した後のＣＦＧを例示する説明図である。

第８６図は変形後の Command Interface のＣＦＧの一部を例示する説明図である。

第８７図は第８６図の続きを示す説明図である。

第８８図は孤立ノードへのＣＦＧの割り当てに関するＣＦＧを例示する説明図である。

第８９図は共有レジスタに関しＣＦＧ生成のためのインライン展開の対象になる記述を例示する説明図である。

第９０図は共有レジスタに関する擬似Ｃ記述を例示する説明図である。

第９１図は第９０図の続を示す説明図である。

発明を実施するための最良の形態

本発明の一例として、実時間制約を有するバス・システムのＪａｖａ（登録商標）によるモデル化及びパラメトリック・モデルチェッキングによる検証とハードウェア合成について説明する。本明細書ではＪａｖａ（登録商標）を単にジャバ言語とも記す。

《設計方法の全体》

第１図には設計方法の全体が示される。設計対象とされるシステムはジャバ言語による記述（ジャバ言語記述）でモデル化される（Ｓ１）。ジャバ言語記述によるモデル化では、単一バス上のデバイスをジャバ言語の run()メソッドを用いて記述する。run()メソッドはマルチスレッドを構成するスレッドにおいて実行させたいプログラムコードがその()内に記述される。クロックはバリア同期を用いて実現される。一般的にバリア同期とは複数のモジュールからのデータを受信する場合に同時刻に処理されるべき全てのデータを待つための同期手法として把握することができる。

次いで、Ｓ１で生成されたジャバ言語記述（ジャバコード）１を読み込み、中間表現に変換する（Ｓ２）。ここで、中間表現２は、コンカレントなコントロール・フロー・グラフ（以下、Ｃ－ＣＦＧ)、コンカレントなパラメータ付き時間オートマトン(以下、Ｃ－ＴＮＦＡ)、パラメータ付き時間オートマトン(以下、ＴＮＦＡ)からなる。ＣＦＧ（コントロール・フロー・グラフ）は一般に関数内部において制御の流れを示すグラムを意味する。ＴＮＦＡ（オートマトン）とは、有限の種類の入力を離散的な時刻に受け、過去から現在までに入力された入力の系列を、回路で決まった個数以下のクラスに分類して記憶し、それに基づいて有限の種類の出力を出す回路の論理的なモデルとして把握することができる。

中間表現２として得られたＴＮＦＡとＲＰＣＴＬ（Real Time Parametric Computation Tree Logic）３で記述された実時間制約を読み込み、パラメトリック・モデル・チェッキングを実行し（Ｓ３）、入力ＲＰＣＴＬ等を満たすパラメータ条件４を導出する。

満足するパラメータ条件がなければ方式変更を行いジャバ言語記述

を修正し、満足するパラメータ条件があれが、そのパラメータ条件をサイクル制約として、Ｃ－ＣＦＧを読み込んで高位合成（Ｓ４）によりＨＤＬ（hardware Description Language）による回路記述５へ変換する。回路記述は例えばＲＴＬ（Register Transfer Level）である。

上記開発方法はそれを実現するためのプログラムをコンピュータ装置で実行することによって行なわれる。ジャバ言語記述からＨＤＬを得る為の開発支援プログラムはディジタル回路の設計支援プログラムとして位置付けることができる。

本設計手法により、単一バスシステムの上流工程でのモデル化、検証が可能となり、かつ実時間制約を満たすハードウェアの設計自動化が可能となる。上流工程での検証が可能なのは、設計対象システムはジャバ言語記述でモデル化され、その記述自身が実行可能にされるからである。

《ジャバ言語によるモデル化に対する制約》

ジャバ言語によるモデル化に際してのジャバ言語記述とバスシステムに対する制約について説明する。単一バス・システムのジャバ言語によるモデル化を目的とするため、ジャバ言語記述に対し、

１）ダイナミック・インスタンシエーションの禁止、

２）run()メソッドからの start()メソッドコールの禁止、

の制約を置く。制約１）はＬＳＩのプロセス変更に関するものであり、ここではハードウェアを扱っている事から、許容可能な制約であると考えられる。また、制約２）はバス・プロトコルの検証を目的とする限りに於いては、直接バス動作に関わる部分のみをモデル化すれば良い為、許容可能な制約であると考えられる。

また、パラメトリック・モデルチェッキング・ツールの制約から、モデルに対して、

３）単一双方向バス・システム、

4）バス権制御は固定優先度

5）バス上の各デバイスは一定周期で処理を終了、の制約を課す。上記制約の緩和は新たな課題として位置付けられる。

《ジャバ言語によるモデル化の為のデザインパターン》

第2図には単一バス・システムのジャバ言語によるモデル化の為のデザインパターンが例示される。単一バス・システムのジャバ言語によるモデル化はＵＭＬ（Unified Modeling Language）で与えられる第2図のデザインパターンに沿って記述される。バス上の各デバイス動作をDeviceImpl Class内のrun()メソッドに実装し、バスを介してアクセスがあるデバイス内レジスタはRegister Class内のattribute（属性）として実装し、バスを介しての同期通信メソッドをRegister Classのメソッドとして実装する。また、バスに対応するBus Class、バスのロック管理を行うBusController Class、及びクロック同期を管理するClock Controller Classを実装する。バスのロック管理とはバスアービトレーションの制御を意味する。第2図の標記において、棒線を付した三角記号△は継承を意味し、例えばDeviceImpl ClassはDevice Classの子クラスであり、DeviceImpl ClassはDevice Classのメソッドを使用可能である。記号→はユース（use）を意味し、例えばDevice ClassはClock Controller Classのメソッドを利用する。

尚、ジャバ言語コードの再利用性を高めるため、下記方針

1）デバイスの追加・削除（DeviceImpl Classの追加・削除）、

2）デバイス動作の変更（DeviceImpl Classのrun()メソッドの変更）、

3）共有変数の追加・削除（Register Classのattributeの追加・削除）、

4）バスプロトコルの追加・変更（Register Classの同期通信メソッドの追加・変更）、

にて、ジャバ言語コードの変更が可能となるようデザイン・パターンを定義している。

第２図のデザインパターンにおける各クラス（Class）について説明する。

（１）Device クラス

デバイスに共通の要素をまとめた抽象クラスである。レジスタやバスは複数ある場合でも処理内容自体が異なることはないため、同じクラスオブジェクトを複数生成ことにより複数存在することを表現できるが、デバイスはデバイスごとにその処理内容が異なる。よって、並列動作を行なうために Thread クラスを継承し、Clock controller Class のインスタンス、Register Class のインスタンス、Bus Class のインスタンスを表すメンバ変数、各デバイスがアクセス可能な共有レジスタの情報を登録する為のメンバ変数、及び各デバイスがアクセス可能な共有レジスタの情報を登録する為のメソッドといったようなデバイスに共通な要素をまとめたクラスとして Device クラスを実装する。

（２）DeviceImpl クラス

実際のデバイスにあたるクラスである。デバイスはデバイスごとに処理内容が異なるため、デバイスに必要な要素をまとめた Device クラスを継承し、その処理内容を記述する。つまり、A という処理を行なう DeviceImplA クラス、B という処理を行なう DeviceImplB クラスといったように処理ごとに Device クラスの実装クラスが存在することになる。

（３）Register クラス

共有レジスタに対応するクラスである。レジスタの値を表すメンバ変数と、その値をリード/ライトするメソッドから成る。

（４）Bus クラス

バスに対応するクラスである。バスが使用中であるか否かの状態を表

すメンバ変数、そのバスに繋がっている共有レジスタをあらわすメンバ変数、及び、状態を変更するメソッドから成る。

（５）BusControllerクラス

バスを介した共有レジスタへのアクセス時のバスロックとバスのロック解放を行なうクラスである。バスのロック・ロック解放はこのクラスに対して依頼する。特に、バスに対して１つデバイスがアクセスを行うとフラグ変数を１に設定するメソッドを含む。このフラグをバスアクセス・フラグとしてメンバ変数として持ち、このメンバ変数により、同一クロック内での複数回のバスロック動作を回避している。

（６）ClockControllerクラス

クロック管理クラスである。バスシステム上の各デバイスにクロック同期動作を行なわせるために、１クロック分の処理終了の通知を集め、全デバイスの処理終了が認められたら、次のクロックの処理を行なってよいという通知を行うと同時にBusController Classのバスアクセス・フラグを０にリセットする。

《モデル化におけるクロック同期メカニズム》

クロック同期メカニズムについて説明する。第３図に示すバリア同期によるクロック同期メソッドを用いて、クロックを実現する。具体的には、１クロック分の処理終了の通知を集め、全デバイスの処理終了が認められたら、次のクロックの処理を行なってよいという通知を行う。

また、バスのロックが残っていない場合には、Bus Classのバスアクセス・フラグを０にリセットする。尚、クロック遷移はパラメトリック・モデル・チェッキングへの入力に際して、パラメータ化される為、必ずしも１クロック単位であるとは限らない。

このクロック遷移のパラメータ化により、サイクル精度でのモデル化よりも抽象度の高いモデル化を実現する。

レジスタへの値書き込みには１クロックを要する為、それを実現する必要がある。これは、consume_1_clock メソッド内で、第４図に例示される Register Class の assignWriteValue メソッドをコールする事で実現している。

《モデル化におけるバス権獲得メカニズム》

バス権管理の同期メカニズムとして、先ずバス権獲得メカニズムについて説明する。バス権獲得メカニズムは、第５図に示すメソッドにより、バス権獲得を管理する。バス権の要求は getBusLock メソッドにより行う。第６図には第５図の各メソッドの呼び出し関係を表すコールグラフが示される。第７図及び第８図には上記バス権獲得メカニズムのジャバ言語記述が例示される。

《モデル化におけるバス権解放メカニズム》

次にバス権解放メカニズムについて説明する。バス権解放メカニズムは第９図にに示すメソッドにより、バス権解放を管理する。バス権の解放は freeBusLock メソッドにより行う。第１０図には第９図の各メソッドの呼び出し関係を表すコールグラフが示される。第１１図には上記バス権解放メカニズムのジャバ言語記述が例示される。

《モデル化における排他的同期リード・メソッド》

バスへの排他的同期アクセス方式として、排他的同期リード・メソッドについて説明する。排他的同期リード・メソッドには以下のシングルリードとバーストリードがある。シングルリードは sync_read メソッドを run()でコールする事で実現される。バーストリードは、sync_burst_read、endBurstAccess、consume_clock メソッドを run()の、synchronized ブロック内で用いる事で実現される。

第１２図には sync_read メソッドのジャバ言語コード、第１３図には run()での sync_read メソッド記述例が示される。run()での記述例に現

れる未定義メソッド、即ち、do_something_w_or_wo_clock_boundary()はクロック境界を含んでもよい何らかの処理を意味し、do_something_wo_clock_boundary()はクロック境界を含まない何らかの処理を意味する。

第１４図には sync_burst_read のジャバ言語コード、第１５図にはendBurstAccess のジャバ言語コード、第１６図には freeBurstBusLockのジャバ言語コード、第１７図には run()でのバーストリードの記述例が示される。バーストリードでは、バスのロックはコールする度に重ねて獲得し、バスの解放を行わないで値を返す。そして、バーストリード回数のリードが終了すると、重ねたロックを一気に解放するという実装方法となっている。この実装により、毎サイクル、リード値が返ってくるというバースト動作を実現している。

《モデル化における排他的同期ライト・メソッド》

バスへの排他的同期アクセス方式として、排他的同期ライト・メソッドについて説明する。排他的同期ライト・メソッドには以下のシングルリードとバーストリードがある。シングルライトは sync_write メソッドを run()でコールする事で実現される。バーストライトは sync_burst_write、endBurstAccess、consume_clock メソッドを run()の、synchronized ブロック内で用いる事で実現される。

第１８図には sync_write のジャバ言語コードが示され、第１９図には run()での sync_write メソッド記述例が示される。

第２０図には sync_burst_write のジャバ言語コードが示され、第２１図には run()でのバーストライトの記述例が示される。バーストライトでは、バスのロックはコールする度に重ねて獲得し、書き込み処理終了後バスの解放を行わないで処理を終了する。そして、バーストライト回数のライトが終了すると、重ねたロックを一気に解放するという実装

方法となっている。この実装により、毎サイクル、ライト値が書き込めるというバースト動作を実現している。

《ジャバ言語記述による実装例》

ジャバ言語記述による実装例を説明する。第２２図には実装例の概略仕様が示される。ここでは、共有メモリ方式の２次元グラフィックス描画・表示システムを一例とする。同図に示されるシステムは、コマンドインタフェース（Command Interface）、ユニファイドメモリ（Unified Memory）、グラフィック描画ユニット（Graphics Rendering Unit）、及び表示ユニット（Display Unit）を有し、それらは双方向バス（Single bi-derection Bus）に共有する。

（１）Command Interfaceは、外部からの描画コマンドを受付け、バスを介して受け付けた描画コマンドをメモリへ転送する。

（２）Unified Memory は、描画コマンド、描画ソースデータ、表示データを一元的に格納するメモリである。システムの低コスト化・部品点数削減に効果大。モデルを単純化する為、Javaでは配列で表現する。本デバイスはスレーブデバイスである。

（３）Graphics Rendering Unitは、Unified Memoryに描画コマンドがあるかをポーリングで調べ、存在する場合は、描画コマンド、描画データをバスを介してリードしながら描画処理を行い、描画結果を内部バッファに格納し、一気に Unified Memoryへバースト転送する。モデルを単純化する為、描画コマンド、描画データ転送は配列への連続アクセスで実現する。

（４）Display Unit は、表示データをリードしながら１ラインずつ表示を行う。モデルを単純化する為、垂直同期はモデル化しない。また、水平帰線期間はバスアクセスを行わないものとする。

上記コマンドインタフェース（Command Interface）、グラフィック

描画ユニット（Graphics Rendering Unit）、及び表示ユニット（Display Unit）がバスマスタデバイスであり、ユニファイドメモリ（Unified Memory）がバススレーブデバイスである。バスマスタデバイスが同時にバス権獲得を行った場合の、バス権獲得の優先順位は、表示ユニット（Display Unit）＞コマンドインタフェース（Command Interface）＞グラフィック描画ユニット（Graphics Rendering Unit）とする。

次に、第２２図の仕様に対する run() method の実装例について説明する。

（１）Command Interface

先ず、コマンドインタフェースに関する run() method の実装例を説明する。その実装例は第２３図乃至第２６図に例示される。外部入力 write_req 信号（具体的には CPU からのライト信号）があり且つ wait 出力信号の値決定に用いる内部変数 wait_flag が false だとコマンド受付を実行（一定サイクル消費）し、その後に先ず wait_flag 変数を true とする。次いで、バス権取得を試み、取得できたら Unified Memory のコマンドフラグ０，１を sync_burst_read で読み出し、値が０（描画コマンド処理済み）であるコマンドフラグに対応するメモリ領域に対して、バス権を取得した状態のままで、sync_burst_write を実行し、コマンドフラグ、描画コマンド、描画ソースデータを転送し、wait_flag 変数を false とする。但し、モデル簡略化の為、例えコマンドフラグが両方とも０であっても、コマンドフラグ０に対応するメモリ領域への sync_burst_write による書き込みのみを実行するものとし、描画ソースデータの転送は省略する。

write_flag 変数が true で、write_req 入力信号が true の場合のみ、wait 信号を true とし、それ以外は wait 信号を false とする。特に、コマンド受付実行中は wait_flag 変数を false とする。尚、wait 信号、

wait_flag 変数ともに初期値は false とする。

また、Command Interface が受け付けるコマンドは、コマンドフラグ、描画コマンド、描画ソースデータ、テクスチャデータの格納開始・終了アドレスからなっており、テクスチャデータの格納開始・終了アドレスは描画コマンドに含まれているものとする。

（２）Unified Memory

ユニファイドメモリに関する run() method の実装例を説明する。その実装例は第２７図に例示される。グラフィックス描画コマンド、描画ソース・グラフィックス・データ、描画後の表示用データを一元的に格納しているメモリ。モデル化を簡略化する為、何も実行しないスレッドとして実現。このメモリの実体は Register Class のインスタンスとして run()メソッド内でアローケートされている。

尚、モデル簡略化の為、メモリ配列を下記

mem_con_reg.current_value[0]：コマンドフラグ０、

mem_con_reg.current_value[1]：描画コマンド０、

mem_con_reg.current_value[2]：コマンドフラグ１、

mem_con_reg.current_value[3]：描画コマンド１、

mem_con_reg.current_value[4]：描画ソース・データ、及び描画後の表示用データ、の構造としている。

実際には、描画ソース・データと描画後の表示データを格納している配列のなすメモリ空間を２分し（一方を Buffer0、他方を Buffer1 とする）、フレーム表示毎にメモリ配列を、

(Buffer0, Buffer1) ＝（描画ソース・データ，描画後の表示用データ）

(Buffer0, Buffer1) ＝（描画後の表示用データ，描画ソース・データ）

となるように、交互に切り替えているとする。従って、描画ソース・データを描画後の表示用データで上書きする事はないとする。更に、モデ

ル簡略化の為、描画ソース・データの転送は省略する。

また、モデル簡略化の為に、テクスチャ・データは省略しており、描画コマンドも最高で２つしか格納しないものとし、コマンド０，１の実行順番がどうであっても描画結果には影響しないものとする。

（３）Graphics Rendering Unit

グラフィック描画ユニットに関する run() method の実装例を説明する。その実装例は第２８図乃至第３２図に例示される。Unified Memory のコマンドフラグ０，１をある一定周期毎に sync_burst_read により読み出す（バス権獲得・解放動作）。何れか一方の値が１であった場合、内部状態変数 render_start を true とし、バス権獲得を試み、獲得に成功したなら、描画コマンド、描画データを sync_burst_read で読み込みながら、描画処理を実行し、描画結果を内部バッファに書き込む（読み込み終了次第、コマンドフラグを０にクリアし、バス権解放する）。但し、モデル簡略化の為、例えコマンドフラグが両方とも０であっても、コマンドフラグ０に対応する描画コマンドのみを実行するものとし、描画ソースデータの転送は省略する。再び、バス権獲得を試み、獲得に成功したなら、描画結果を sync_burst_write で Unified Memory へ転送し、バス権を解放し、内部状態変数 render_start を false とする。尚、内部状態変数 render_start の初期値は false である。

（４）Display Unit

表示ユニットに関する run() method の実装例を説明する。その実装例は第３３図及び第３４図に例示される。Unified Memory から描画後の表示用データを読み込みＣＲＴへ出力する。但し、モデル簡略化の為、水平同期のみを扱うものとする。これは、水平帰線期間内に描画が開始・終了する為のサイクル制約をパラメトリック・モデルチェッキングで求める事を目的としているからである。

さて、Display Unit は、内部変数 display_start を ture に設定し、Unified Memory から描画後の表示用データを sync_burst_read にて読み込む。データ読み込み終了後に内部変数 display_start を false として、ライン表示を実行し、ライン表示終了後から水平帰線期間に相当する適当なサイクルを消費し、再び、ライン表示を同じ手順で行うものとする。また、転送サイクルのみをモデル化すれば良いので、メモリアドレスは簡略的に表現し、常に同一メモリ空間から読み出しているものとする。

《パラメトリック解析》

ここでパラメトリック解析の目標について説明する。本来、描画は１フレーム表示の表示期間及び、垂直帰線期間の間に１フレーム分の実行を終了しておれば良いと考えられるので、水平帰線期間での描画が満たすべき性質を勘案すると、検証内容としては下記条件を満たすサイクル制約の導出で十分である。即ち、「表示用データ取得終了後に描画を開始し、次の表示開始までに描画を終了する事がある。但し、描画開始・終了にデータ転送サイクルは含まれ、表示はｎサイクル以内に終了する。」である。上記制約をＲＰＣＴＬ(Real Time Parametric Computation Tree Logic) で記述すると下記

EF{<display_end>(AF(<render_begin>true)
&&{AF(<render_end>true)AU(<display_begin>true)})}
&& AG{[display_begin](AF<=n([display_end]true))}

の３個の制約のアンド条件となる。尚、表示データ取得終了迄に未実行の描画コマンドが Unified Memory 上に常に存在する事が仮定できるものとする。ここで、n は適当な正整数であり、各変数は下記の通り、

display_begin：内部変数 display_start の０から１への立ち上がりイベント

display_end：内部変数 display_start の１から０への立ち下がりイベント

render_begin：内部変数 render_start の０から１への立ち上がりイベント

render_end：内部変数 render_start の１から０への立ち下がりイベント

を表す。

パラメトリック・モデルチェッキングを高速化する目的で、ＲＰＣＴＬを

AG{[display_begin](AF<=n([display_end]true))}　（＊）

と、それ以外の部分に分離し、例えば、

EF{<display_end>(EF(<render_begin>true)&&{EF(<render_end>true)AU(<display_begin>true)})}

から実行し、パラメータ条件を導出し、導出されたパラメータから（＊）のｎの上限を求め、既に求めたパラメータ条件に矛盾しないパラメータ条件が導出できるまで、ｎの値を減じて繰り返し、（＊）のパラメトリック解析を実施するものとする。

ＲＰＣＴＬやそれを用いたパラメトリック・モデルチェッキングの詳細は次の文献、“Akio Nakata and Teruo Higashino: ”Deriving Parameter Conditions for Periodic Timed Automata Satisfying Real-Time Temporal Logic Formulas”, Proc. of IFIP TC6/WG6.1 Int'l Conf. on Formal Techniques for Networked and Distributed Systems(FORTE2001), Cheju Island, Korea, Kluwer Academic Publishers, pp.151-166, Aug. 2001.”に記載ある。

《中間表現への変換》

第３５図には中間表現へ変換工程の詳細が全体的に例示される。バス

システムをモデル化して記述されたジャバ言語記述はC(concurrent)－ＣＦＧに変換され（Java2C-CFG）、Ｃ－ＣＦＧはＣ－ＴＮＦＡに変換され（C-CFG2C-TNFA）、Ｃ－ＴＮＦＡはＴＮＦＡに変換される（C-TNFA2TNFA）。パラメトリック・モデルチェッキングを経てＣ－ＣＦＧがＨＤＬに変換される（C-CFG2HDL）。尚、上記Java2C-CFG等の標記において、数字“2”は“to”を意味するものと理解されたい。

《Java２C-CFG》

ジャバ言語記述からＣ－ＣＦＧへの変換アルゴリズムの概要を説明する。各 run() メソッドに対応するＣＦＧを生成する。特に、クロック同期メソッド（consume_clock）はクロック境界として識別し、呼び出しメソッドは呼び出し関係を表すノードを用いて表現する。各 run() メソッドに対応するＣＦＧが生成されると、fork ノードを設け、そのノードから各ＣＦＧの開始ノードへの fork 枝を付加する。特に、バス上の各デバイスに run()メソッドが対応するものとして Java が記述されている事を前提とし、メモリデバイスに関しては、ＣＦＧ生成対象外である事の指定を与えるものとする。

次いで、呼び出しメソッドのＣＦＧを作成し、呼び出しメソッドが synchronized ブロック内に存在する場合は、呼び出しメソッド内の synchronized をＣＦＧから削除する。特に、呼び出しメソッドが Register クラス内の通信メソッドである場合はＣＦＧの作成を行わず、インスタンス関係からどのデバイス内のどのレジスタへのアクセスかと通信メソッドの名前だけを識別し、その情報をＣＦＧに保持する。但し、図ではどのレジスタへのアクセスかの情報は省略している。尚、通信メソッドが内部にクロック同期メソッドを含む場合は、その出力枝にクロック境界をマークする。更に、呼び出しメソッド全体のサイクル制約を導出する以外は、呼び出しメソッドをインライニングする。この例

では、Rendersing メソッド、Display メソッドはインライニングしない。

synchronized を予め定めたＣＦＧに展開し、ハード合成用の固定優先度スケジューラをＦＳＭ（Finite State Machine）の形式でスケルトンとして予め与えてあるライブラリと実際にＣＦＧを生成した run() メソッドの個数を用いて、自動生成する。ここで、ＣＦＧはハード合成用とパラメトリック・モデルチェッキング用の２種作成する。

尚、固定値伝播や、代入によるローカル変数消去等のコードレベル最適化をＣＦＧ上で実施する。特に、信号の入力・出力・入出力はメンバ変数で表し、端子方向に関しては、プログラム内での代入文で右辺にあるのか左辺にあるのかや条件文での変数の用いられ方を解析して決定する。入出力信号として識別された場合、データ依存性を考慮して、入力と出力に適当な信号のリネームを行って分離する。

Java２C-CFG に関し、以下で、Java 記述例に沿って、各 run()メソッドのＣＦＧ生成結果、Ｃ－ＣＦＧ生成結果、及び固定優先度・スケジューラのＦＳＭについて説明する。

先ずＣＦＧの形式について説明する。第３６図にはＣ－ＣＦＧの形式が例示される。各ＣＦＧは分岐枝の開始と終了を表すノードと、ループ枝の開始と終了を表すノードを含み、分岐条件、ループ終了条件を対応する枝に付加した形式とする。特に、クロック境界ノードを枝上に付加する事でクロック境界を表現する。また、メソッドコールに対応するノードを用いる事で、サブＣＦＧとのリンクを表現する。各ＣＦＧの並列実行を表す fork ノードはＣＦＧへの頂点へ付加する事で表現する。

synchronized の扱い（ハード合成用）について説明する。第３７図に例示されるように、synchronized の開始をＣＦＧ上で begin_sync とラベル付けされたノードとして表し、synchronized の終了を end_sync とラベル付けされたノードでし、ＣＦＧを生成。その後、その両ノード

を夫々下図に示すように変換する。

synchronized の扱い（パラメトリック・モデルチェッキング用）について説明する。第３８図に例示されるように、synchronized の開始をＣＦＧ上で Begin_sync とラベル付けされたノードとして表し、synchronized の終了を End_sync とラベル付けされたノードでし、ＣＦＧを生成。その後、その両ノードを夫々第３８図に示すように変換する。

上記に従うと、前記 Command Interface の場合、ＣＦＧは第３９図に示される。getWriteSignal()メソッドは、シミュレーションを実施する為に行った記述であり、処理系への入力では外部からの入力信号 write_req であるとして修正されているものとし、入力テストベクタの個数を表す dcom_index を補正する記述も削除されているものとする。また、drawing_commands も配列として表現されているが、これもシミュレーションを実施する為になされたものであり、内部変数 input_command に入力変数 drawing_commands が入力されている記述が処理系に入力されているのもとして扱う。

前記 Command Interface に関し、ハード合成用のＣＦＧは第４０図に示され、パラメトリック・モデルチェッキング用のＣＦＧは第４１図に例示される。

前記グラフィック描画ユニット（Graphics Rendering Unit）に関し、第４２図及び第４３図にはそのＣＦＧが例示され、第４４図及び第４５図にはそのパラメトリック・モデルチェッキング用のＣＦＧが例示される。

前記表示ユニット（Display Unit）に関し、第４６図にはそのＣＦＧが例示され、第４７図にはそのパラメトリック・モデルチェッキング用のＣＦＧが例示される。

第４８図にはコマンドインタフェース（Command Interface）のＣＦ

G、グラフィック描画ユニット（Graphics Rendering Unit）のＣＦＧ、及び前記表示ユニット（Display Unit）のＣＦＧにおける夫々の開始ノードの結合状態が例示される。

固定優先度スケジューラ（Fixed Priority Scheduler）について説明する。実際にトップレベルのＣＦＧを生成したrunメソッドの数を識別し、夫々のデバイスにバス権取得通知を行うための信号locked_iの組みからなるステートを作成する。単一バスシステムであるため、それらステートは、１つのlocked_iのみが１でその他が０であるものを構成すれば良い。また、全てのlocked_iが０であるステートも作成する。優先度情報に基づき、バス権要求信号lock_iを受けて行う状態遷移枝を設ければバス権管理用の固定優先度スケジューラが構成できる。特に、全てのlocked_iが０であるステートをリセット解除時の開始ステートとする。また、各状態遷移は1<=t<=klの時間制約が付加されているものとする。このパラメータは後のパラメトリック・モデルチェッキングで決定されるパラメータである。尚、例えば、kl=2とし、各遷移を２クロック遷移とした場合は、出力値は変化が起こるまで、前の値を保持しているものとして解釈するものとする。

第４９図には本例に対応する固定優先度スケジューラが例示される。同図における信号の意味は、

lock_1：Command Interfaceからのバス権要求信号

lock_2：Graphics Rendering Unitからのバス権要求信号

lock_3：Display Unitからのバス権要求信号

locked_1：Command Interfaceへのバス権取得通知信号

locked_2：Graphics Rendering Unitへのバス権取得通知信号

locked_3：Display Unitへのバス権取得通知信号

である。尚、バス権取得の優先度が、

Display Unit ＞ Command Interface ＞ Graphics Rendering Unit

であるので、遷移条件は、

lock_3 : Display Unit への遷移

!lock_3&&lock_1 : Command Interface への遷移条件

!lock_3&&!lock_1&&lock_2 : Graphics Rendering Unit への遷移条件

otherwise : 全ての locked_i が０であるステートへの遷移条件

となる。

《Abstraction of each CFG》

夫々のＣＦＧの抽象化処理について説明する。先ず、そのアルゴリズムの概要を説明する。各 BasicBlock に対して、0<=t<=ki なる時間遷移条件を付加し、Begin_sync ノードと End_sync ノードに対して 0<=t<=kl を付加し、通信メソッドを表すノードに対して、バースト開始に 1<=t<=kb1、バースト動作に 0<=t<=kb2、バースト終了に 0<=t<=kb3 を付加する。但し、通信メソッドを表すノードに対しては、バースト開始が２サイクル以上、バースト動作時は１サイクル以上、バースト終了が１サイクル以上を要すると考え、後述のノードのマージによる抽象化を行う際には、サイクル制約の補正を行う。Begin_sync と End_sync ノード以外の制約を付加したノードをクロック境界と見做して抽象化を行う。

通信メソッドは動作内容を全て抽象化し、連続する場合は、１つのノードにまとめ、例えばバースト開始(1<=t<=kb1)、クロック境界(t=1)、(バースト動作(0<=t<=kb2)＋クロック境界(t=1)) ３回、バースト終了(0<=t<=kb3)、クロック境界(t=1)とすると、

6<=t<=(kb1+1-1)+3(kb2+1-1)+(kb3+1-1) = kb1+3kb2+kb3 (>=2+3*1+1 = 6)

なるサイクル制約を持つクロック境界として１つにまとめる。また、分

岐ノードは全て非決定分岐であるとする。Basic Blockノードに関しては、入力RPCTLで用いる変数のみ残し他は抽象化する。分岐ノード下位に存在するBasic Blockに対応する部分が異なる分岐で同じ場合は、1つを残して他を削除する。

特にループノードに関しては、break、continueを含まない固定回数ループの場合はアンローリングを実施し、それ以外の場合は、ループを抜ける条件枝の直前に全体で 0<=t<=kloop のサイクル制約を持つようなＣＦＧノードの集合を設ける事でループを削除する。ここで、kloopの満たすべき条件を別に算出するものとする。更に連続するクロック境界はパーコレーション・ベースの移動を行う事で、１つに纏め、遷移サイクル制約を更新する。

ここで、各クロック境界のパラメータ化により、クロック境界に与えた遷移サイクル制約の意味は、クロック境界から次のクロック境界への遷移に対するサイクル制約である。

コマンドインタフェース（Command Interface）のＣＦＧに対する抽象化の様子が順を追って第５０図乃至第５６図に例示される。第５７図にはグラフィック描画ユニット（Graphics Rendering Unit）のＣＦＧに対する抽象化処理の結果が例示される。第５８図には表示ユニット（Display Unit）のＣＦＧに対する抽象化処理の結果が例示される。

《C-CFG2C-TNFA》

Ｃ－ＴＮＦＡからＴＮＦＡへの変換処理について説明する。ＴＮＦＡとは時間オートマトンであり、各遷移が、

s-a@?t[P(t)]->s'

の形式で表されるものである。ここで、各記号の意味は、

s,s'：状態、

a：動作(省略可能)、

@?t:状態ｓを訪れてから動作ａを実行するまでの経過時間をｔに代入、
P(t)：時間制約(tに関する線形不等式の論理結合）、
である。また、その意味は「ｓからｔ単位時間経過後に状態 s'に遷移。ただし、ｔは時間制約P(t)を満たす場合に限る」である。

Ｃ－ＴＮＦＡは、複数のＴＮＦＡが並列動作するモデルであり、動作 sync は高々１つのＴＮＦＡしか実行できないモデルである。特に連続する動作 sync に対しては他のＴＮＦＡが割り込めない。尚、各ＴＮＦＡに対して、初期状態から初期状態への遷移に対して、上限を与える為の時間制約を課す事が可能である。

次にＣ－ＣＦＧからＣ－ＴＮＦＡへの変換処理について説明する。先ず、その変換アルゴリズムの概要を説明する。Begin_sync、End_sync で挟まれた部分を識別し、sync ブロックとする。Sync ブロック内に存在しないクロック境界ノードにステート割り当て候補とし、また sync ブロック内であっても検証性質に必要となる信号がBasic Blockとして与えられている場合はその前後のクロック境界ノードにステートを割り当て候補とし、最後に Begin_sync、End_sync の変換で導入したクロック境界をステート割り当て候補とする。得られたステート割り当て候補を重複が無いようにし、ステート割り当てを行い、各ステートからステートまでをＤＦＳ（Depth First Search）でトラバースする事でＴＮＦＡへの変換を行う。得られたＴＮＦＡに対して、sync ブロックに対応する遷移を検出し、sync 遷移とする。連続する sync 遷移でない遷移は１つの遷移に纏める事でステート数の削減を行う。

コマンドインタフェース（Command Interface）のＣＦＧに対するＴＮＦＡへの変換の様子が順を追って第５９図乃至第６１図に例示される。

第６２図乃至第６５図にはグラフィック描画ユニット（Graphics

Rendering Unit）のＣＦＧに対するＴＮＦＡへの変換の様子が順を追って例示される。第６５図において、検証性質は、「表示用データ取得終了後に描画を開始し、次の表示開始までに描画を終了する事がある。但し、描画開始・終了にデータ転送サイクルは含まれ、表示はｎサイクル以内に終了する。」であり、これは、描画が開始される事がもしあればという前提にたった検証性質であるから、Graphics Rendering Unitに対応するＴＮＦＡにおいて、表示データ取得終了迄に未実行の描画コマンドがUnified Memory上に存在しない事を表すR2からR1への遷移はこの検証には不要となる。従って、これは削除されている。

第６６図乃至第６８図には表示ユニット（Display Unit）のＣＦＧに対するＴＮＦＡへの変換の様子が順を追って例示される。

《C-TNFA2TNFA》

Ｃ－ＴＮＦＡからＴＮＦＡへの変換アルゴリズムの概要を説明する。先ず、下記方針（１）～（５）に従ってＣ－ＴＮＦＡから積ＴＮＦＡを構成する。

（１）sync 動作とそれ以外の動作は互いにインタリーブされる。ここで、sync動作を行っている間にバス権をとる必要が無い動作、即ちsync でない動作を（一般に複数）並行して実行できることを表現。但し、sync動作を１遷移実行する間の他のTNFAの並行遷移に遷移枝を通る回数に制限を設け、それを満たす各遷移の遷移時間の上限を導出する。特に、回数は１から始め、意味のある解が得られるまで、回数を１づつインクリメントするものとする。

（２）複数のsync動作が実行可能な場合は静的優先度が最も高いものによって遷移する。ここで、動いていない状態ｓに関してはage(s,t)（状態ｓで時間ｔだけが経過した状態）に遷移。ただし、連続するsync遷移の間には他の動作は割り込めない。

（３）age(s,t)以降の遷移に関しては、時間ｔに関して下記補正を行う。即ち、s -[P(t1)]-> s'ならばage(s,t) -[P(t+t1)]->s'[t+t1/t1]、とする。ここで、s[e/t] ： 状態ｓからの遷移条件に現れる変数ｔを式ｅで置き換えることを表す。尚、再構成したＰ(t)が矛盾した式になっている場合、その遷移は削除し、その遷移へのみ到達する状態全てを削除するものとする。

（４）各遷移の時間変数ｔは下記に示すように、異なる名前に変更する。即ち、s-[P(t)]->s'-[Q(t)]->s"は s-[P(t1)]->s'-[Q(t2)]->s"に変更する。age(s,t)に関する補正によって、各遷移条件はそれ以前の遷移の時間変数も含む式になるため、それらを区別するために、この変更が必要となる。

（５）上限を与えたＴＮＦＡがある場合、構成後のＴＮＦＡでその初期状態を含む状態間の遷移がその上限を満たさない場合、その途中にある状態で上限を満たす遷移内に存在する状態を削除対象外とし、削除対象外とならなかった状態を削除する。

尚、Abstraction of each CFGのステップで抽象化対象から除外されたＲＰＣＴＬに用いられている変数を伴う遷移が現れるとその次の状態まで構成し、上記構成方針での作成での切りの良いところで、Abstraction of TNFAをコールし、構成中のＴＮＦＡの抽象化を行う。

次にＣ－ＴＮＦＡからＴＮＦＡへの変換処理を詳細に説明する。先ず、遷移回数の仮定による遷移時間の上限を決定する。即ち、各ＴＮＦＡの強連結成分を求め、強連結成分内での遷移枝を１回のみ通る各遷移の下限遷移時間の総計が最大となる最長パスと、そのパス上の頂点でその後段の頂点への遷移時間の下限が最小の頂点を求め、最長パスにその後段頂点へのパスを追加したパスの２つを求める。次いで、夫々のパスの下限遷移時間の総計を求める。これにより、

Command Interface : 22 23

Graphics Rendering Unit : 28 29

Display Unit : 11 12

が求まる。さて、各ＴＮＦＡの２つ目の下限遷移時間の総計から１引いた値、

Command Interface : 22

Graphics Rendering Unit : 28

Display Unit : 11

を求める。各ＴＮＦＡの各遷移の上限時間を、他のＴＮＦＡに対して求めた値の最小値を与える事で決定し、得られた時間遷移の上限値が矛盾したものとなっていないかを、既に上限が与えられている遷移の上限値と比較する事で調べ、もし、矛盾がでたなら２回に増加させる。そうでなければ、上限が与えられていない遷移に求めた上限を与える事で線形不等式の論理結合を構成する。特に、回数を増加させる場合、最長パスの下限遷移時間の総計と回数の積を取った値と、その値に先に求めた最小の遷移時間を加えた値を算出すればよい。

尚、パラメトリック・モデルチェッキングを実施し、パラメータ条件が非負整数解を持たない場合にも回数の増加を行って処理をすすめる事とする。ここで、非負整数解の算出には、線形計画法を用いる。具体的には、フリーソフト LP-SOLV を用いて得られたパラメータ条件を満たす範囲で、パラメータの和が最小となる解を算出する。仮に、得られた解が意味をなさない場合は、パラメータの最小値を付加するなどして対処する。

さて、本例の場合、遷移枝を一回のみ遷移可能とした仮定では、各ＴＮＦＡの遷移時間の上限として

Command Interface : 11

Graphics Rendering Unit : 11

Display Unit : 22

が求まる。また、矛盾が起こることなく、以下に示す線形不等式の論理結合、

0<=k1+k2+kloop1<=7 && 5<=kb1+kb2+kb3+kl<=8 && 7<=kb1+3kb2+kb3+kl<=11&& 1<=kl<=11 && 1<=k1+k2+k3+kloop1+kl<=6 && 3<=k4<=11 && 5<=kb1+kb2+kb3+kl<=11 && 6<=kb1+4kb2+kb3+3kr1<=12 && 1<=3kr2+kl<=8&& 9<=kb1+5kb2+kb3+kl<=12 && kr1=kb2-1 && 8<=kb1+5kb2+kb3<=24 && 1<=kl+6kd<=19

が得られる。また、遷移枝を２回まで遷移可能とした場合、各ＴＮＦＡの遷移時間の上限として

Command Interface : 22

Graphics Rendering Unit : 22

Display Unit : 44

が求まり、以下に示す線形不等式の論理結合、

0<=k1+k2+kloop1<=18 && 5<=kb1+kb2+kb3+kl<=19 && 7<=kb1+3kb2+kb3+kl<=22 && 1<=kl<=22 && 1<=k1+k2+k3+kloop1+kl<=17 && 3<=k4<=2 && 5<=kb1+kb2+kb3+kl<=22 && 6<=kb1+4kb2+kb3+3kr1<=23 && 1<=3kr2+kl<=19 && 9<=kb1+5kb2+kb3+kl<=23 && kr1=kb2-1 && 8<=kb1+5kb2+kb3<=46 && 1<=kl+6kd<=41

が得られる。

この工程は、Assume Guarantee Reasoning と呼ばれる手法であり、パラメトリック解析での実施は公知ではない。

第６９図には第６１図、第６４図、第６８図で求めたＴＮＦＡの積ＴＮＦＡの構成例が示される。これに基づいてＴＮＦＡを求める過程が第７０図乃至第７６図に示される。

《Abstraction of ＴＮＦＡ》

ＴＮＦＡの抽象化処理について説明する。先ず、そのアルゴリズムの概要を説明する。ＴＮＦＡで抽象化時に残した変数への代入を行っている遷移枝と、その遷移枝の始点ノード及び終点ノードのみを残し、残すべき頂点を始点として、次に残すべき頂点が現れるまで頂点と辺をトラバースしながら遷移時間を再計算する事で、その他の頂点を全て抽象化する。但し、葉に対応する状態は抽象化の対象から除外する。

先の例では抽象化の実施がまだ起こらない段階までしかＴＮＦＡを構成しなかったので、抽象化が必要となる様先の例を意図的に若干修正して、第７７図乃至第７９図にに C-TNFA2TNFA で Abstraction of TNFA を適時コールした場合の処理の経過の一部を示す。ここでは、kl には具体的な値は与えないものとする。

《パラメトリック解析結果》

前記文献に述べられているアルゴリズムを用いて、遷移を２回以上通らない制約で、探索深さ最大値１６として、以下の検証性質

EF (<displayend>((AF(<renderbegin>true))
and ((AF(<renderend>true)) AU (<displaybegin>true))))

に関してパラメタ条件を導出すると、以下の条件、

0 <= kd && 0 <= kr2 && 0 <= kr1 && 0 <= kb3 && 0 <= kb2 && 0 <= kb1 && 0 <= kloop1 && 0 <= k5 && 4 <= k4 && 0 <= k3 && 0 <= k1 && 0 <= k2 && 12 <= kb1+9kb2+kb3 && 4 <= kl)

を得る。これと先に求めた、遷移を２回以上通らないという制約から得られたパラメータ条件

0<=k1+k2+kloop1<=7 && 5<=kb1+kb2+kb3+kl<=8 && 7<=kb1+3kb2+kb3+kl<=11 && 1<=kl<=11 && 1<=k1+k2+k3+kloop1+kl<=6 && 3<=k4<=11 && 5<=kb1+kb2+kb3+kl<=11 && 6<=kb1+4kb2+kb3+3kr1<=12

&& 1<=3kr2+kl<=8 && 9<=kb1+5kb2+kb3+kl<=12 && kr1=kb2-1 && 8<=kb1+5kb2+kb3<=24 && 1<=kl+6kd<=19

との論理積を取って、得られたパラメータ条件に対して、目的関数を各パラメタ値の合計として、これを最小化するという線形計画問題をフリーソフト LP_SOLVE にて解かせると、第８０図の解を得る。

第８０図の結果では、バスアクセス終了通知が０サイクルで行われなければならず、また、描画サイクル kr2 と表示サイクル kd が０であるので、意味のある解とは言えないので、kb1=2,kb2=1,kb3=1 の制約を追加し、第８１図の解を得た。

第８１図の結果でも、描画サイクル kr2 と表示サイクル kd が０であるので、意味のある解とは言えないので、kb1=2,kb2=1,kb3=1 の制約に加えてさらに、kr1 以外のパラメタ変数は 1 以上という制約を追加し、第８２図の解を得た。以後、このパラメータ値を採用して議論を進める。

《ハード合成》

ハードウェア合成処理についてその概要を先ず説明する。（１）～（１１）の方針にてハードウェアの生成が行なわれる。

（１）事前に Register クラスに登録された通信メソッドの種類からバスコマンドの割り当てを行う。

（２）ユーザ情報として、どのデバイスにどの共有レジスタが存在するかを受付け、各デバイス内のレジスタ数に対応するデバイス内アドレスを割り当てる。

（３）共有レジスタが割付られたデバイス（run()メソッド）の数を取得し、共有レジスタが割り当てられたデバイスにグローバルなアドレスを割り当てる。

（４）インライニング（展開）を実施しなかった通信メソッド以外のメソッドのインライニングを行う。（通信メソッドと指定したメソッド

以外のメソッドは事前にインライニングされている事に注意。）

（５）パラメトリック解析結果から得られた、Basic Blockの実行サイクルを各デバイスの合成用ＣＦＧに反映する。

（６）各デバイス夫々のＣＦＧを変形し、通信メソッドとそれ以外のコントロールフロー部が互いに通信するモデルに変換する。

（７）前記（６）で得られた変換後のＣＦＧ内の通信メソッドに対応する部分に、バスコマンド生成記述を挿入する。通信メソッドは、グローバルアドレス、共有レジスタアドレス、バスコマンドを出力し、データ読み出し・書き込みの何れかを実行する記述からなるＣＦＧとなり、それらの信号生成・受理に要するサイクルは、パラメトリック解析の結果により決定される。

（８）共有レジスタアドレスから、各デバイス内のレジスタの配列インデックスを生成するアドレスデコーダとグローバル・アドレスを識別するアドレスデコーダを共有レジスタが割り当てられたデバイス内に生成し、出力トライステートを共有レジスタの出力に接続し、トライステート・イネーブル信号として、各デバイスからのデータ入力ステージを表す信号の論理和を用いる 。共有レジスタの入力に関しては、取り込み動作が起こらない限り内部変数は前の値を保持する為バスから信号を接続すれば良よく、各デバイスからのデータ入力ステージを表す信号の論理和を元にデータ取り込み判定を行う。但し、これらはＣＦＧとして表現する。 （自デバイス内に共有レジスタがあったとしても、この方式に従って、アクセスが行われているものとする。）

（９）変形等で得られた各ＣＦＧの信号通信関係（入力、出力）を識別し（バスは入出力）、夫々をモジュールとして識別する。

（１０）各ＣＦＧを「サイクルアキュレートなプログラム記述からのハード合成」に従って、ＨＤＬへと変換する。

（１１）バスに対してリピータ回路をＨＤＬの形式で挿入する。

前記バスコマンドの割り当てに関し、本例で登録したバスアクセス・メソッドは、sync_read、sync_write、sync_burst_read、sync_burst_write、endBurstAccessであるが、各run()メソッドのＣＦＧを解析すると、実際に用いられているバスアクセスメソッドは、sync_burst_read、sync_burst_write、endBurstAccessであるため、バスコマンドを

sync_burst_read 2'b00

sync_burst_write 2'b01

endBurstAccess 2'b10

NOP 2'b11

のように割り当てる。

上記アドレス割り当てに関しては、共有レジスタは Unified Memory にのみ割り付けられ、Unified Memory の仕様から割り付けられた共有レジスタは

mem_con_reg.current_value[0]：コマンドフラグ０、

mem_con_reg.current_value[1]：描画コマンド０、

mem_con_reg.current_value[2]：コマンドフラグ１、

mem_con_reg.current_value[3]：描画コマンド１、

mem_con_reg.current_value[4]：描画ソース・データ、及び描画後の表示用データ、

mem_con_reg.current_value[5]：描画ソース・データ、及び描画後の表示用データ、

mem_con_reg.current_value[6]：描画ソース・データ、及び描画後の表示用データ、

mem_con_reg.current_value[7]：描画ソース・データ、及び描画後の表

示用データ、

mem_con_reg.current_value[8]：描画ソース・データ、及び描画後の表示用データ、

mem_con_reg.current_value[9]：描画ソース・データ、及び描画後の表示用データ、となる。尚、ユーザ情報から各レジスタのビット幅が指定されているものとし、それぞれ３２ビット（unsigned int）であるとする。

バスアクセス・メソッドはレジスタ内のバイトアクセス等を行わないので、各レジスタのアドレスを

mem_con_reg.current_value[0]：4'b0000、

mem_con_reg.current_value[1]：4'b0001、

mem_con_reg.current_value[2]：4'b0010、

mem_con_reg.current_value[3]：4'b0011、

mem_con_reg.current_value[4]：4'b0100、

mem_con_reg.current_value[5]：4'b0101、

mem_con_reg.current_value[6]：4'b0110、

mem_con_reg.current_value[7]：4'b0111、

mem_con_reg.current_value[8]：4'b1000、

mem_con_reg.current_value[9]：4'b1001、のように決定する。

また、アドレス割り当てに関し、共有レジスタが割り当てられたデバイスはUnified Memoryしか存在しないため、グローバルアドレスの割付を行わない。

尚、本例では１つのデバイスのみに共有レジスタの割り当てを行っているが、以降の説明に於いてこれは一般性を欠くものではない。それは、以降に処理手順を示す事で明らかとなる。

前記BasicBlockへの実行サイクルの割り当てに関しては、ここでは、

Command Interface のみ取り上げて説明を行う。特に、先に求めたパラメータ値から、kl = 4, kb1 = 2, kb2 = 1, kb3 = 1、k1 = k2 = k3 = 1、として説明を行う。パラメータ条件から、各 BasicBlock の下に割り当てられたサイクル数のクロック境界を挿入する。ここでは、バスアクセス・メソッドを扱う対象としていない事に注意。更に、与えたパラメータで、例を示しているが一般性を失わない事に注意。次ページに変形後のＣＦＧを示す。尚、kl の値は、直接各デバイスのハード合成用ＣＦＧには影響を与えない事にも注意。これは、固定優先度スケジューラにのみ影響する。第８３図には BasicBlock への実行サイクルの割り当ての例が示される。

固定優先度スケジューラの修正に関しては、既に得られているＦＳＭ (Finite State Machine) の遷移サイクルを kl に変更する、即ち、クロック境界を各遷移枝に４つ付加すれば良い。ここからのＨＤＬ生成は、各状態からの遷移を分岐ノードに表現し直し、かつ各状態での信号代入はその状態への遷移枝上での最後のクロック境界直下に BasicBlock を設け、そこにレジスタ代入文を記述する形式で置き換える事で、このＦＭＳ自体をＣＦＧと見做す事で、「サイクルアキュレートな記述からのＨＤＬ生成」を用いる事で可能である。第８４図に示す固定優先度スケジューラの一部を用いて変形後のＣＦＧが第８５図に示される。

ＣＦＧの変形に関しては、元のＣＦＧからバスアクセス・メソッドの括りだしを行い、バスアクセス・メソッドと元のＣＦＧとの通信ノードを設る。具体的には、下記を行う。

１）ハード合成用のＣＦＧ生成段階で、クロック境界を含むバスアクセス・メソッドのクロックをバスアクセス・メソッドを表すノードの直下にクロック境界を追加したが、これを削除する。

２）バスアクセス・メソッドを表すノードを下記処理を行う Basic

Blockに置き換える。

① 出力信号 start_comm に１を代入する。

② バスコマンドを出力信号 bus_cmd に代入する。バースト転送の場合、初期サイクルでは、

<1> 出力信号 init の０を代入

<2> アクセスするデバイスのアドレスと共有レジスタのアドレスを連結して出力信号 address に代入

<3> クロック境界挿入×ｎｂｆ

<4> リードメソッドなら、代入すべき変数に入力信号 data_in を代入

<5> ライトメソッドなら、出力すべき値または変数を出力信号 data_out に代入

<6> クロック境界×ｍｂｆ、を行う。ここで、ｎｂｆ＋ｍｂｆ＝ｋｂ１である。この例では、ｎｂｆ＝ｍｂｆ＝１とする。

バースト転送の場合、転送終了では、

<1>クロック境界挿入×ｎ、を行う。ここで、ｎ＝ｋｂ３である。ここでは、ｎ＝１とする。

バースト転送の場合、それ以外では、

<1> 出力信号 init に１を代入

<2> アクセスするデバイスのアドレスと共有レジスタのアドレスを連結して出力信号 address に代入

<3> クロック境界挿入×ｎｂ

<4> リードメソッドなら、代入すべき変数に入力信号 data_in を代入

<5> ライトメソッドなら、出力すべき値または変数を出力信号 data_out に代入

<6> クロック境界×ｍｂを行う。ここで、ｎｂ＋ｍｂ＝ｋｂ２である。この例では、ｎｂ＝０、ｍｂ＝１とする。

シングル転送の場合、

<1> アクセスするデバイスのアドレスと共有レジスタのアドレスを連結して出力信号 address に代入

<2> クロック境界挿入×ｎｓ

<3> リードメソッドなら、代入すべき変数に入力信号 data_in を代入

<4> ライトメソッドなら、出力すべき値または変数を出力信号 data_out に代入

<5> クロック境界挿入×ｍｓ、を行う。ここで、ｎｓ＋ｍｓ＝ｋｓである。

③ 出力信号 start_comm に０を代入する。

３）バスアクセスメソッドを表す孤立ノードを作成し、置き換えて生成した Basic Block との通信ノードを設ける。通信ノードとして、

start_comm : Basic Block → 孤立ノード

bus_cmd : Basic Block → 孤立ノード

address : Basic Block → 孤立ノード

data_in : 孤立ノード → Basic Block

data_out : Basic Block → 孤立ノード、を設ける。

第８６図及び第８７図には変形後の Command Interface のＣＦＧの一部が示される。第８７図において、クロック境界の傍らに記載された変数はパラメータであり、その数だけクロック境界を生成することを意味する。

孤立ノードへのＣＦＧの割り当てに関しては、第８８図のＣＦＧを自動生成する。第８８図において、特に、AD_BUS はアドレスバスを、D_BUS はデータバスを表し、data_in_en_i はデータ入力ステージを、data_out_en_i はデータ出力ステージを表す。ここで、i は各デバイスのインデックスを表す。また、アドレスバスのバス幅は、割り当てたバ

スコマンドのビット幅とアドレスのビット幅の合計とし、データバスはアクセスするレジスタのビット幅とする。また、クロック境界の傍にかかれた変数はパラメータであり、その数だけクロック境界を生成する事を意味する。尚、各変数の値は、ＣＦＧの変形過程で用いた変数の値と等しい。

共有レジスタに関しては、第８９図の記述をインライン展開したものに対応するＣＦＧの生成を行えばよい。但し、入出力変数である AD_BUS、D_BUS の入力、出力への分離は行わず、またＣＦＧ上での最適化に於いては、AD_BUS、D_BUS に対する変数最適化は実施しないものとする。第８９図、第９０図及び第９１図には共有レジスタに関する擬似Ｃ記述である。サイクルアキュレートな記述からのＨＤＬ生成に関しては本発明者による先の出願（特願２００２－３００７３）を参照することができる。

以上本発明者によってなされた発明を実施形態に基づいて具体的に説明したが、本発明はそれに限定されるものではなく、その要旨を逸脱しない範囲において種々変更可能であることは言うまでもない。

例えば、積オートマトンの構成に関しては、ＴＮＦＡ積オートマトンを構成するとき、遷移枝の通過回数で Assume Guarantee Reasoning を実施したが、本来は、sync 動作の通過回数で行うべきである。理由は、sync 動作はバスアクセス動作を意味しており、並列に動いているＴＮＦＡが何回 sync 動作を実施したかは、そのＴＮＦＡに対応するデバイスがバスアクセスを何回行ったかに対応するからであり、Refinement 及びパラメトリック・モデルチェッキングで得られるのは、遷移サイクルの条件のみならず、その検証性質を実施している間に各デバイスが高々どれだけバスアクセスを行うかの情報も得られるからである。但し、この手法であると、組み合わせ爆発を起し得るので、何らかの高速な枝

切りが必要となる。

積オートマトンの構成に関しては、今回は、パラメトリック・モデルチェッキングの停止性を吟味せず、バウンデッド・モデルチェッキングとして、検証性質を満たす必要条件を求める事を行ったが、本来は十分を求める必要がある。その為、アルゴリズムの停止性に関する研究を更に行う必要がある。

バスモデルの拡張に関しては、単一双方向バスのみではなく、単方向バスや、ローカルバスを含むもの、またバス・ブリッジを介してバスが階層化されているような複数バスシステムを扱うようにしてもよい。

また、水平帰線期間のみを扱ったが、本来は垂直帰線期間もモデル化して、1フレームを表示している間に最低1フレーム分の描画が終了するための十分条件を求める必要がある。

産業上の利用可能性

本発明は、並列動作を記述できる言語からディジタル回路のハードウェア合成を行うデータ処理システムなど広く適用することができる。

請 求 の 範 囲

1．並列動作の記述が可能なプログラム言語を用いて複数のデバイスを定義したプログラム記述を入力し、入力したプログラム記述を中間表現に変換し、この中間表現に対し、実時間制約を満足するパラメータを生成し、生成したパラメータに基づいてハードウェア記述言語による回路記述を合成することを特徴とするシステム開発方法。

2．前記中間表現は、コンカレントなコントロールフローフラグ、コンカレントなパラメータ付き時間オートマトン、又はパラメータ付き時間オートマトンであることを特徴とする請求項1記載のシステム開発方法。

3．前記パラメータ生成に、パラメトリック・モデルチェッキングを行うことを特徴とする請求項2記載のシステム開発方法。

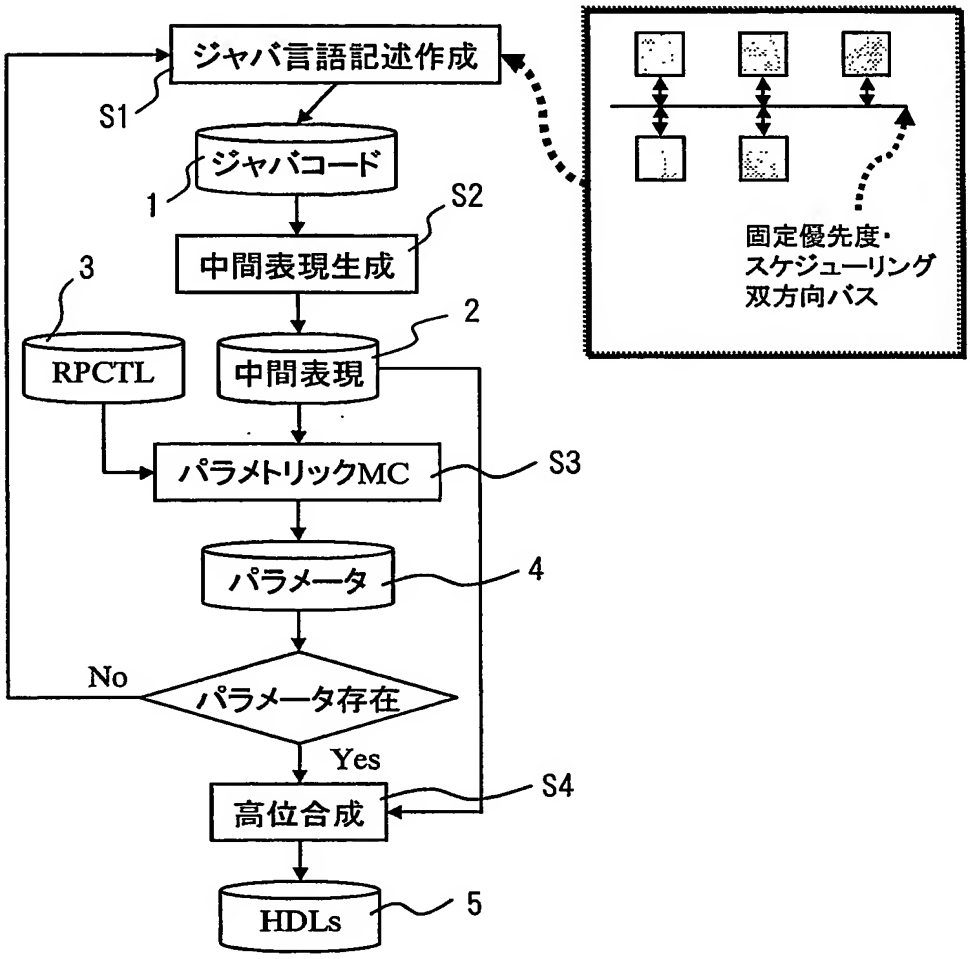
4．前記実時間制約はＲＰＣＴＬで与えられることを特徴とする請求項3記載のシステム開発方法。

5．前記プログラム記述はｒｕｎメソッドを用いてデバイスの定義を行い、バリア同期を用いてデバイスのクロック同期を定義することを特徴とする請求項4記載のシステム開発方法。

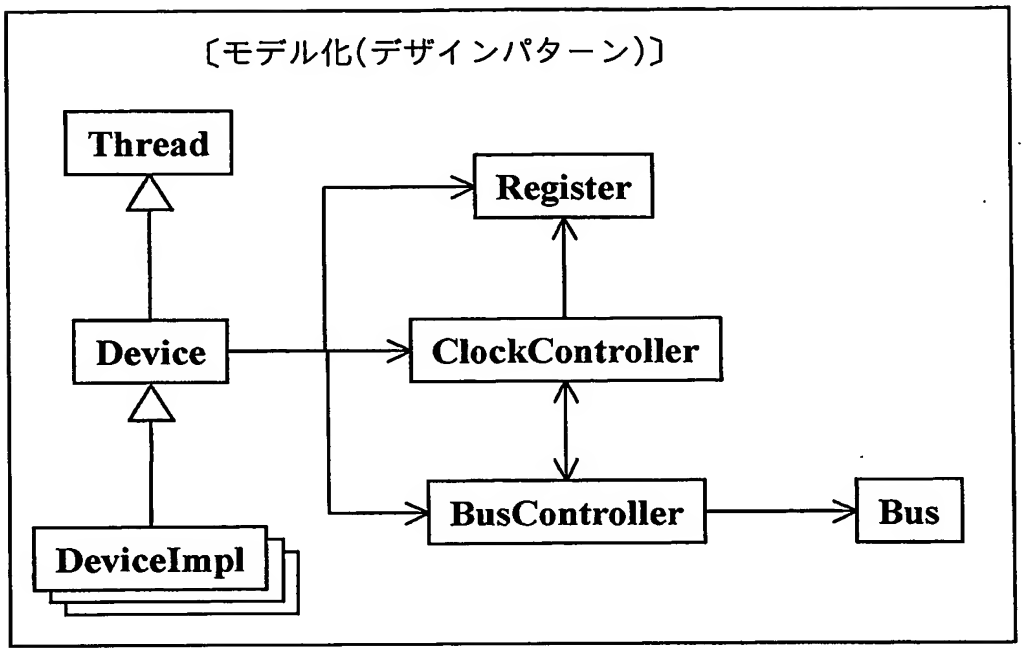
6．並列動作の記述が可能なプログラム言語を用いて複数のデバイスを定義したプログラム記述を入力し、入力したプログラム記述を中間表現に変換し、この中間表現に対し、実時間制約を満足するパラメータを生成し、生成したパラメータに基づいてハードウェア記述言語による回路記述を合成することを特徴とするデータ処理システム。

7．前記プログラム記述はｒｕｎメソッドを用いてデバイスの定義を行い、バリア同期を用いてデバイスのクロック同期を定義することを特徴とする請求項6記載のデータ処理システム。

第1図


ジャバ言語記述作成
S1
ジャバコード
1
S2
中間表現生成
3
RPCTL
中間表現
2
パラメトリックMC
S3
パラメータ
4
No
パラメータ存在
Yes
S4
高位合成
HDLs
5
固定優先度・
スケジューリング
双方向バス

第２図


〔モデル化(デザインパターン)〕
Thread
Register
Device
ClockController
BusController
Bus
DeviceImpl

第３図

〔モデル化(クロック同期)〕

```
private void consume_1_clock() {
  /* 全デバイス数とこのmethodを実行したデバイスの数が等しいかチェック */
  if (++this.current_num == this.device_num) {
    this.current_num = 0;
    for (int i=0; i<this.device_num; i++) {
      /* レジスタ代入の実行 */
      registers[i].assignWriteValue();
    }
    /* バスロックを識別するフラグ変数(バスアクセス・フラグ)の初期化 */
    if (this.bc.getBusyCount() == 0) {
      this.bc.initLockDoneOnceFlag();
    }
    notifyAll();
  } else {
    try {
      wait();
    } catch (InterruptedException e) {
    }
  }
}
```

第４図

〔モデル化(クロック同期)〕

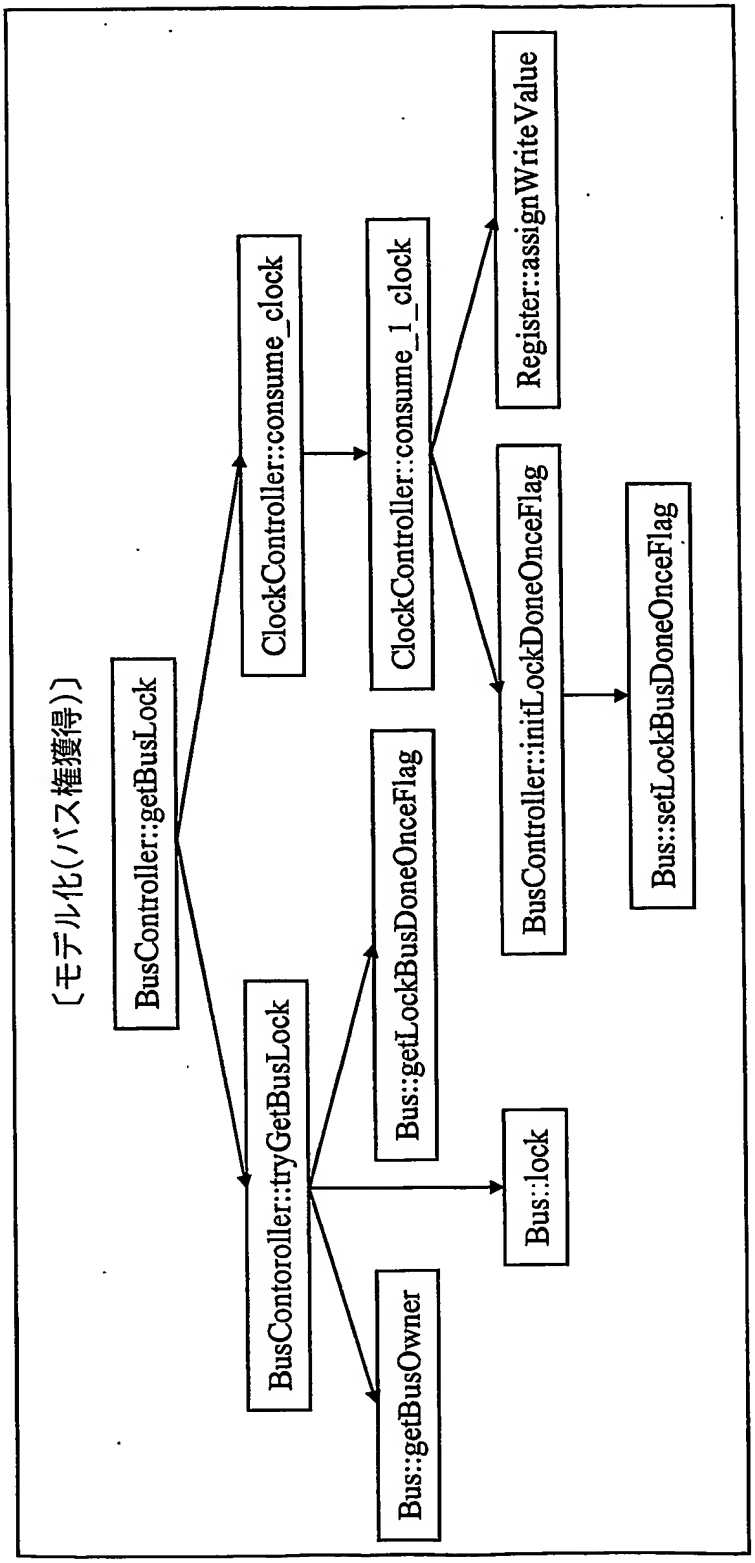
```
public void assignWriteValue() {
  /* 後述のsync_writeメソッド又は、sync_burst_writeメソッドにより
     レジスタへの書き込みが実行がなされたかを判定 */
  if (this.write_access) {
    /* 実際に書き込みを行ったレジスタ(配列index)への書き込みを
       実行 */
    this.current_value[this.update_index] = this.write_value;
    /* ライト・アクセス・フラグをリセット */
    this.write_access = false;
  }
}
```

第5図

〔モデル化(バス権獲得)〕

method名	
getBusLock	tryGetBusLockがtrueを返せば、lockメソッドをコールする事でバス権を獲得し、falseの場合、consume_clockをコールし、clock_numの数だけクロック消費。
tryGetBusLock	getBusOwnerがnullで且つLockBusDoneOnceFlagがfalseなら、lockをコールし、getBusLockにtrueを返す。それ以外の場合はfalseを返す。
getBusOwner	現在バスをロックしているthreadのオブジェクトを返す。ロックしているオブジェクトが無ければnullを返す。
lock	現在、Bus classオブジェクト(lockメソッド)を実行しているThreadを、現在バスをロックしているオブジェクトとして登録する。
getBusDoneOnceFlag	バスアクセス・フラグの値を取得する。
consume_clock	consume_1_clockをコールする。
consume_1_clock	クロック同期メカニズムで既に説明済み。
initLockDoneOnceFlag	引数をfalseとしてsetLockBusDoneOnceFlagをコール。
setLockBusDoneOnceFlag	引数をバスアクセス・フラグに代入。
assignWriteValue	共有レジスタへのライトアクセスがあった場合、次のクロックへ進む前にレジスタ代入を実行する。

第６図


〔モデル化(バス権獲得)〕
BusController::getBusLock
BusContorolller::tryGetBusLock
Bus::getBusOwner
Bus::lock
Bus::getLockBusDoneOnceFlag
ClockController::consume_clock
ClockController::consume_1_clock
BusController::initLockDoneOnceFlag
Register::assignWriteValue
Bus::setLockBusDoneOnceFlag

第７図

〔モデル化(バス権獲得)〕

```
public void getBusLock(int clock_num) {
  /* バス権が獲得出来たかチェック */
  while (this.tryGetBusLock() == false) {
    /* バス権が獲得出来なかったので、clock_numの数だけクロック消費 */
    cc.consume_clock(clock_num);
  }
}
```

第８図

〔モデル化(バス権獲得)〕

```
private synchronized boolean tryGetBusLock() {
  /* 現在バスがロックされておらず、且つそのクロック内で一度もバスが
     ロックされた事がない(即ちバスアクセス・フラグがfalse)か、をチェック */
  if ((bus.getBusOwner() == null) && (bus.getLockDoneOnceFlag() == false)) {
    /* バスをロック */
    this.bus.lock();
    /* ロック回数をインクリメント */
    this.busycount++;
    /* バス権が獲得できた事をtrueとして返す */
    return true;
  } else if (bus.getBusOwner() == Thread.currentThread()) {
    /* 現在バスをロックしているスレッドによりさらにロック要求があった場合、
       重ねてロックする。そのためにロック回数をインクリメント */
    this.busycount++;
    /* バス権が獲得できた事をtrueとして返す */
    return true;
  } else {
    /* バス権が獲得出来なかった事をfalseとして返す */
    return false;
  }
}
```

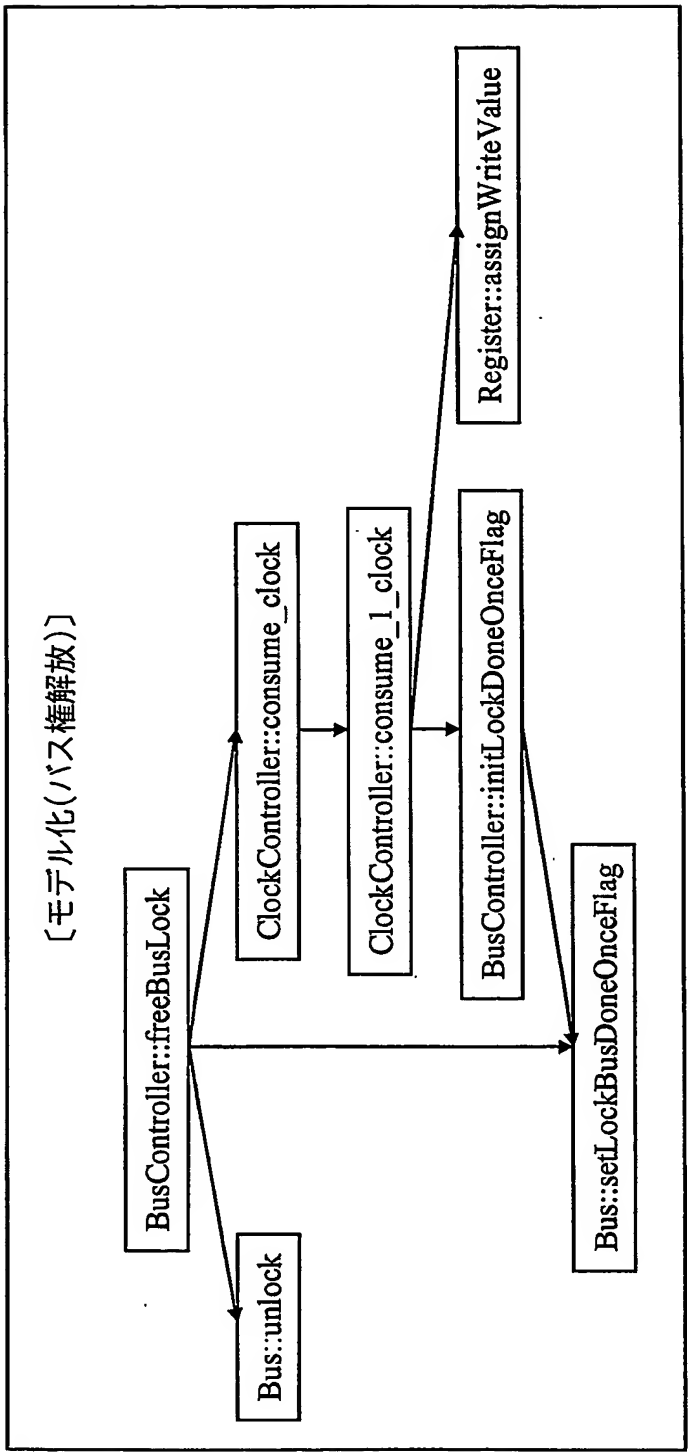
第９図

〔モデル化(バス権開放)〕

method名	
freeBusLock	unlockメソッドをコールする事でバス権を開放し、trueを引数としてsetLockBusDoneOnceFlagをコールし、consume_clockをコールし、clock_numの数だけクロック消費。
unlock	現在バスをロックしているオブジェクトが、現在Bus classオブジェクト(lockメソッド)を実行しているThreadかを確認し、そうであれば、現在バスをロックしているオブジェクトをnullとする。
consume_clock	consume_1_clockをコールする。
consume_1_clock	クロック同期メカニズムで既に説明済み。
initLockDoneOnceFlag	引数をfalseとしてsetLockBusDoneOnceFlagをコール。
setLockBusDoneOnceFlag	引数をバスアクセス・フラグに代入。
assignWriteValue	共有レジスタへのライトアクセスがあった場合、次のクロックへ進む前にレジスタ代入を実行する。

第１０図

〔モデル化(バス権解放)〕



第１１図

〔モデル化(バス権解放)〕

```
public void freeBusLock(int clock_num) {
   synchronized (this) {
      /* コールしたスレッドがバスをロックしているスレッドかチェック */
      if (this.bus.getBusOwner() == Thread.currentThread()) {
         /* ロック回数をデクリメント */
         this.busycount--;
         /* デクリメントした結果が0かどうかチェック */
         if (this.busycount == 0) {
            /* バスのロックを解放 */
            this.bus.unlock();
         }
         /* バスアクセス・フラグをtrueに設定 */
         this.bus.setLockDoneOnceFlag(true);
      }
   }
   /* clock_numの数だけクロック消費 */
   cc.consume_clock(clock_num);
}
```

第１２図

〔モデル化(排他的同期リード)〕

```
public synchronized int sync_read(BusController bc,
                                  int index,
                                  int clock_num) {
   /* バス権獲得 */
   bc.getBusLock(clock_num);
   /* 指定したオブジェクト共有変数の値を読み込む */
   int read_value = this.current_value[index];
   /* バス権解放 */
   bc.freeBusLock(clock_num);
   /* 読み込んだ値を返す */
   return read_value;
}
```

〔sync_read メソッド〕

第１３図

〔モデル化(排他的同期リード)〕

```
public void run() {
  Register other_r0 = (Register)super.access_registers.get(0);
  int  read_value;
  while (true) {
    this.do_something_w_or_wo_clock_boundary1();
    lread_value = other_r0.sync_read(super.bc0, 1);
    this.do_something_w_or_wo_clock_boundary2();
}
```

〔run()での記述例〕

第１４図

〔モデル化(排他的同期リード)〕

```
public int sync_burst_read(BusController bc, int index,
                                    int clock_num) {
   /* コールされる度にロック条件を満たすならロックを重ねる */
   bc.getBusLock(clock_num);
   /* 指定したオブジェクト共有変数の値を読み込む  */
   int read_value = this.current_value[index];
   /* 読み込んだ値を返す */
   return read_value;
}
```

〔sync_burst_read メソッド〕

第15図

〔モデル化(排他的同期リード)〕

```
public void endBurstAccess(BusController bc, int clock_num) {
   bc.freeBurstBusLock(clock_num);
}
```

〔endBurstAccess メソッド〕

第16図

〔モデル化(排他的同期リード)〕

```
public void freeBurstBusLock(int clock_num) {
   synchronized (this) {
     /* コールしたスレッドがバスをロックしているスレッドかチェック */
      if (this.bus.getBusOwner() == Thread.currentThread()) {
         /* ロック回数を0に戻す */
         this.busycount = 0;
       /* バスのロックを解放 */
         this.bus.unlock();
      }
     /* バスアクセス・フラグをtrueに設定 */
      this.bus.setLockDoneOnceFlag(true);
   }
  /* clock_numの数だけクロック消費 */
   cc.consume_clock(clock_num);
}
```

〔 freeBurstBusLock メソッド〕

第17図

〔モデル化(排他的同期リード)〕

```
public void run() {
  Register other_r1 = (Register)super.access_registers.get(1);
  int  read_value[10]
  while (true) {
    this.do_something_w_or_wo_clock_boundary1();
    /* バースト・リード(10回連続リード) */
    synchronized (this) {
      int i;
      for (i=0; i<10; i++) {
        read_value[i] = other_r1.sync_burst_read(super.bc, i, 1));
        super.cc.consume_clock(1);
        this.do_something_wo_clock_boundary1();
      }
      other_r1.sync_burst_read(super.bc, i, 1);
      other_r1.endBurstAccess(super.bc, 1);
      this.do_something_wo_clock_boundary2();
    }
    this.do_something_w_or_wo_clock_boundary2();
  }
}
```

〔run()での記述例〕

第１８図

〔モデル化(排他的同期ライト)〕

```
public synchronized void sync_write(BusController bc,
                                    int write_value,
                                    int index, int clock_num) {
  /* バス権獲得 */
  bc.getBusLock(clock_num);
  /* 指定したオブジェクト共有変数への書き込み値を保持 */
  this.write_value  = write_value;
  /* アクセスした配列を通知 */
  this.update_index = index;
  /* 共有変数への書き込みがあった事を通知
     (consume_1_clockにて次クロックへの遷移
     直前に共有変数に書き込みを実施) */
  this.write_access = true;
  /* バス権解放 */
  bc.freeBusLock(clock_num);
}
```

〔sync_write メソッド〕

第１９図

〔モデル化(排他的同期ライト)〕

```
public void run() {
  Register other_r0 = (Register)super.access_registers.get(0);
  int  write_value;
  while (true) {
    this.do_something_w_or_wo_clock_boundary1();
    other_r0.sync_write(super.bc, write_value0, 1);
    this.do_something_w_or_wo_clock_boundary2();
  }
```

〔run()での記述例〕

第２０図

〔モデル化(排他的同期ライト)〕

```
public int sync_burst_read {(BusController bc, int write_value, int index, int clock_num) {
   /* コールされる度にロック条件を満たすならロックを重ねる */
   bc.getBusLock(clock_num);
   /* 指定したオブジェクト共有変数への書き込み値を保持 */
   this.write_value  = write_value;
   /* 共有変数への書き込みがあった事を通知
     (consume_1_clockにて次クロックへの遷移
      直前に共有変数に書き込みを実施) */
   this.write_access = true;
}
```

sync_burst_write メソッド

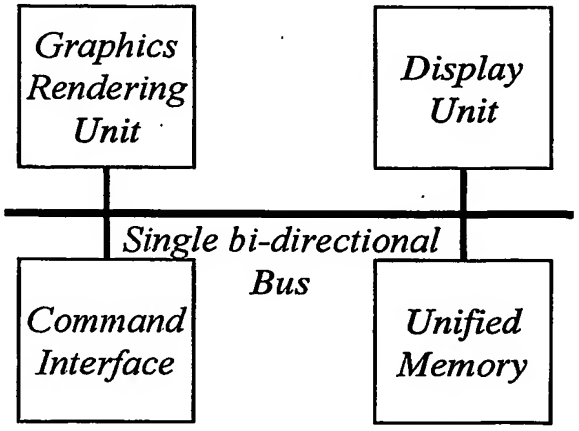
第２１図

〔モデル化(排他的同期ライト)〕

```
public void run() {
  Register other_r0 = (Register)super.access_registers.get(0);
  int  write_value[10]
  while (true) {
    this.do_something_w_or_wo_clock_boundary1();
    /* バースト・ライト(10回連続ライト) */
    synchronized (this) {
      int i;
      for (i=0; i<10; i++) {
        other_r0.sync_burst_write(super.bc, write_value[i], 1));
        super.cc.consume_clock(1);
        this.do_something_wo_clock_boundary1();
      }
      other_r1.sync_burst_write(super.bc, write_value[i], 1);
      other_r1.endBurstAccess(super.bc, 1);
      this.do_something_wo_clock_boundary2();
    }
    this.do_something_w_or_wo_clock_boundary2();
  }
}
```

run()での記述例

第２２図



第２３図

〔run() methodの実装例(Command Interface)〕

```
public void run() {
  Register mem_con_reg = (Register)super.access_registers.get(2);
  int[] drawing_commands = {100, 101, 102, 103, 104};
  int     dcom_index = 0;
  int     com_flag_0 = 0;
  int     com_flag_1 = 0;
  while (true) {
    /* 外部入力信号write_req信号があり、且つwait_flag信号がfalseのとき
       コマンド受け付けを実行。それ以外の時は、単にクロックを消費するだけ。 */
    if (this.getWriteReqSignal() && (this.wait_flag == false)) {
      /* コマンドの受付を実行。 */
      this.input_command = drawing_commands[dcom_index++];
      /* 一定サイクル消費。ここでは3クロックとした。*/
      super.cc.consume_clock(3);
      /* wait_flag変数をtrueにセット */
      this.wait_flag = true;
      /* write_req信号がありwait_flag変数がtrueなので、
         wait出力信号をtrueにセットし、1クロック消費。 */
      this.wait_output_signal = true;
      super.cc.consume_clock( 1);
```

シミュレーション用に
挿入した記述

第２４図

〔run() methodの実装例(Command Interface)〕

```
/* コマンドフラグの値が0の場合に対応する描画コマンドの値を更新する。
   0,1共にコマンドフラグの値が0の場合は、0に対応するほうを優先する。
   0,1共にコマンドフラグの値が1の場合は、一定クロック待機して再度実行。 */
while (true) {
  synchronized (this) {
    /* コマンドフラグ0、1をバーストリードする。 */
    com_flag_0 = mem_con_reg.sync_burst_read(super.bc, 0, 1);
    super.cc.consume_clock(1); //クロック消費。
    com_flag_1 = mem_con_reg.sync_burst_read(super.bc, 2, 1);
    super.cc.consume_clock(1); //クロック消費。
    if (com_flag_0 == 0) {
      /* コマンドフラグ0の値が0の場合。または、コマンドフラグ0,1の値が共に0の場合。*/
      /* 描画コマンド0に入力されたコマンドを書き込む。*/
      mem_con_reg.sync_burst_write(super.bc, input_command, 1, 1);
      super.cc.consume_clock(1);
      /* コマンドフラグ0の値を1にする。 */
      mem_con_reg.sync_burst_write(super.bc, 1, 0, 1);
      /* バーストモードの終了(クロック消費含む)。 */
      mem_con_reg.endBurstAccess(super.bc, 1);
      break;
```

第２５図

〔run() methodの実装例 (Command Interface)〕

```
            } else if (com_flag_1 == 0) {
                /* コマンドフラグ1の値が0の場合。*/
                /* 描画コマンド1に入力されたコマンドを書き込む。*/
                mem_con_reg.sync_burst_write(super.bc, input_command, 3, 1);
                super.cc.consume_clock(1);
                /* コマンドフラグ1の値を1にする。*/
                mem_con_reg.sync_burst_write(super.bc, 1, 2, 1);
                /* バーストモードの終了(クロック消費含む)。*/
                mem_con_reg.endBurstAccess(super.bc, 1);
                break;
            } else {
                /* 0,1共にコマンドフラグの値が1の場合。*/
                /* バーストモードの終了(クロック消費含む)。*/
                mem_con_reg.endBurstAccess(super.bc, 1);
                /* 一定クロック待機。ここでは3クロック待機させた。*/
                super.cc.consume_clock(3);
            }
        } // end of synchronized
    } // end of nested while-loop
```

第２７図

〔run() methodの実装例 (Unified Memory)〕

```
public void run() {
  /* Unified Memoryへの書き込みを行うバス上のデバイス内の
     レジスタをインスタンス化 */
  // Graphics Rendering Unit 内のレジスタ群
  Register   renderer_reg = (Register)super.access_registers.get(0);
  // Display Unit 内のレジスタ群
  Register   display_reg = (Register)super.access_registers.get(1);
  // Command Interface 内のレジスタ群
  Register   com_fetch_reg = (Register)super.access_registers.get(2);
  /* モデル簡略化の為、実際には何も行わない */
  while (true) {
    /* 1クロック消費。*/
    super.cc.consume_clock(1);
  }
}
```

第２６図

〔run() methodの実装例(Command Interface)〕

```
        /* wait_flag変数をfalseにセット。 */
        this.wait_flag = false;
        /* wait_flag変数がfalseになったので、wait出力信号をfalseに戻し、1クロック消費。*/
        this.wait_output_signal = false;
        super.cc.consume_clock(1);
        /* 描画コマンド配列をを最後まで使用した場合はそのインデックスを最初に戻す。*/
        if (dcom_index == drawing_commands.length) {
          dcom_index = 0;
        }
      } else {
        super.cc.consume_clock(1);
      }
    } // end of while-loop
}
```

シミュレーション用に挿入した記述

第２８図

〔run() methodの実装例 (Graphics Rendering Unit)〕

```
public void run() {
   Register   mem_con_reg = (Register)super.access_registers.get(2);
   int[] rendering_result = new int[6];
   int    current_command = 0;
   int          read_data = 0;
   int        com_flag_0 = 0;
   int        com_flag_1 = 0;
   while (true) {
   /* 待ち状態。*/
   super.cc.consume_clock(3);
      while (true) {
         synchronized (this) {
            /* コマンドフラグ０、１をバーストリードする。*/
            com_flag_0 = mem_con_reg.sync_burst_read(super.bc, 0, 1);
            super.cc.consume_clock(1); //クロック消費。
            com_flag_1 = mem_con_reg.sync_burst_read(super.bc, 2, 1);
            /* バーストモードの終了（クロック消費含む）。*/
            mem_con_reg.endBurstAccess(super.bc, 1);
         } // end of synchronized
```

第２９図

〔run() methodの実装例(Graphics Rendering Unit)〕

```
if (com_flag_0 == 1) {
  /* コマンドフラグ0の値が0の場合。または、コマンドフラグ0,1の値が共に0の場合。*/
  synchronized(this) {
    /* 描画コマンド0の値を読み込む。*/
    current_command = mem_con_reg.sync_burst_read(super.bc, 1, 1);
    /* render_startをtrueに。*/
    this.render_start = true;
    super.cc.consume_clock(1);
    for (int i=0; i<3; i++) {
      /* データの読み出し。*/
      read_data = mem_con_reg.sync_burst_read(super.bc, i+4, 1);
      /* クロック消費。*/
      super.cc.consume_clock(1);
      /* レンダリング。*/
      rendering_result[i] = this.rendering(read_data, current_command);
    } // end of for-loop
    /* コマンドフラグ0の値を0にする。*/
    mem_con_reg.sync_burst_write(super.bc, 0, 0, 1);
    /* バーストモードの終了(クロック消費含む)。*/
    mem_con_reg.endBurstAccess(super.bc, 1);
  } // end of synchronized
```

第３０図

〔run() methodの実装例 (Graphics Rendering Unit)〕

```
        for (int i=3; i<6; i++) {
          /* レンダリング。*/
          rendering_result[i] = this.rendering(read_data, current_command);
          /* クロック消費。*/
          super.cc.consume_clock(1);
        } // end of for-loop
        break;
      } else if (com_flag_1 == 1) {
        /* コマンドフラグ１の値が0の場合。*/
        synchronized(this) {
          /* 描画コマンド1の値を読み込む。*/
          current_command = mem_con_reg.sync_burst_read(super.bc, 3, 1);
          /* render_startをtrueに。*/
          this.render_start = true;
          super.cc.consume_clock(1);
          for (int i=0; i<3; i++) {
            /* データの読み出し。*/
            read_data = mem_con_reg.sync_burst_read(super.bc, i+4, 1);
            /* クロック消費。*/
            super.cc.consume_clock(1);
            /* レンダリング。*/
            rendering_result[i] = this.rendering(read_data, current_command);
          } // end of for-loop
```

第３１図

〔run() methodの実装例 (Graphics Rendering Unit)〕

```
            /* コマンドフラグ1の値を0にする。*/
            mem_con_reg.sync_burst_write(super.bc, 0, 2, 1);
            /* バーストモードの終了(クロック消費含む)。*/
            mem_con_reg.endBurstAccess(super.bc, 1);
          } // end of synchronized
          for (int i=3; i<6; i++) {
            /* レンダリング。*/
            rendering_result[i] = this.rendering(read_data, current_command);
            /* クロック消費。*/
            super.cc.consume_clock(1);
          } // end of for-loop
          break;
        } else {
          /* 0,1共にコマンドフラグの値が1の場合。*/
          /* 一定クロック待機。ここでは3クロック待機させた。*/
          super.cc.consume_clock(3);
        }
      } // end of nested while-loop
```

第３２図

〔run() methodの実装例 (Graphics Rendering Unit)〕

```
    /* レンダリングされたデータをメモリに書き込む。*/
    synchronized (this) {
      for (int i=0; i<6; i++) {
        if (i == 5) {
          /* データの書き込み。*/
          mem_con_reg.sync_burst_write(super.bc, rendering_result[i], i+4, 1);
          /* バーストモードの終了(クロック消費含む)。*/
          mem_con_reg.endBurstAccess(super.bc, 1);
        } else {
          /* データの書き込み。*/
          mem_con_reg.sync_burst_write(super.bc, rendering_result[i], i+4, 1);
          /* クロック消費。*/
          super.cc.consume_clock(1);
        }
      } // end of for-loop
    } // end of synchronied
    /* レンダリング終了。*/
    this.render_start = false;
  } // end of while-loop
}
```

第３３図

〔run() methodの実装例 (Display Unit)〕

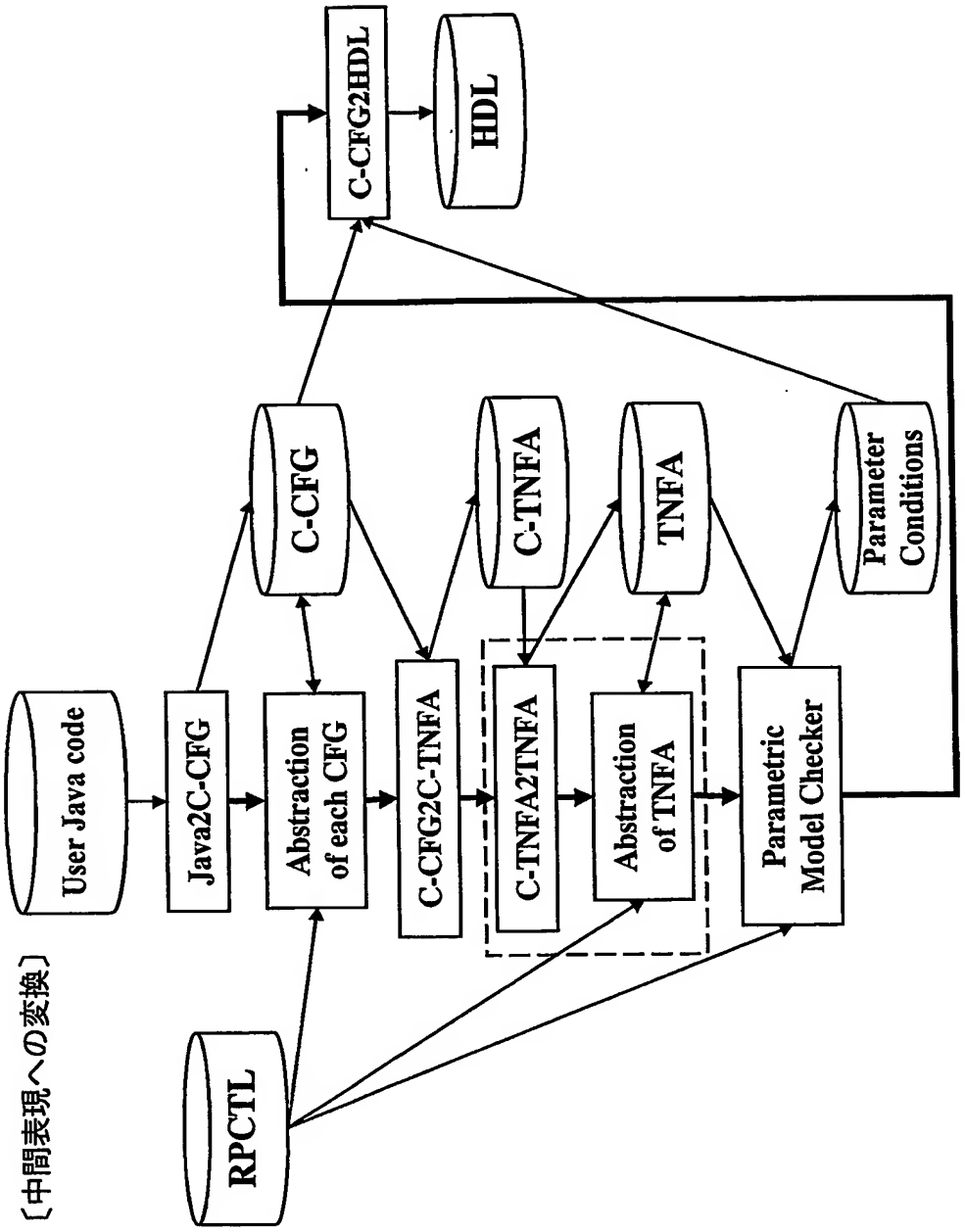
```
public void run() {
  Register mem_con_reg = (Register)super.access_registers.get(2);
  int read_data = 0;
  while (true) {
    synchronized (this) {
      for (int i=0; i<6; i++) {
        if (i == 5 ) {
          /* データの読み出し。*/
          read_data = mem_con_reg.sync_burst_read(super.bc, i+4, 1);
          /* バーストモードの終了(クロック消費含む)。*/
          mem_con_reg.endBurstAccess(super.bc, 1);
        } else {
          /* データの読み出し。*/
          read_data = mem_con_reg.sync_burst_read(super.bc, i+4, 1);
          /* 表示の為のデータロード開始。*/
          this.display_start = true;
          /* クロック消費。*/
          super.cc.consume_clock(1);
        }
      } // end of for-loop
    } // end of synchronized
```

第３４図

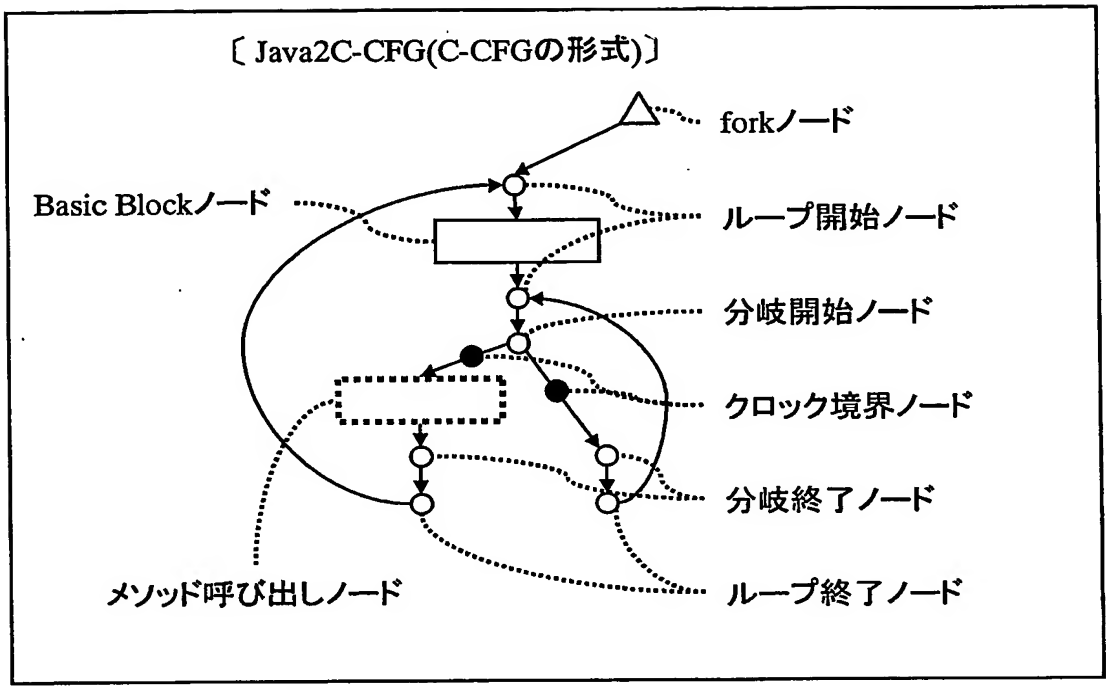
〔run() methodの実装例 (Display Unit)〕

```
      /* 表示の為のデータロード終了。*/
      this.display_start = false;
      /* 表示。*/
      for (int i=0; i<6; i++) {
        this.display(read_data);
      }
      /* 待ち。*/
      super.cc.consume_clock(3);
      }
    }
}
```

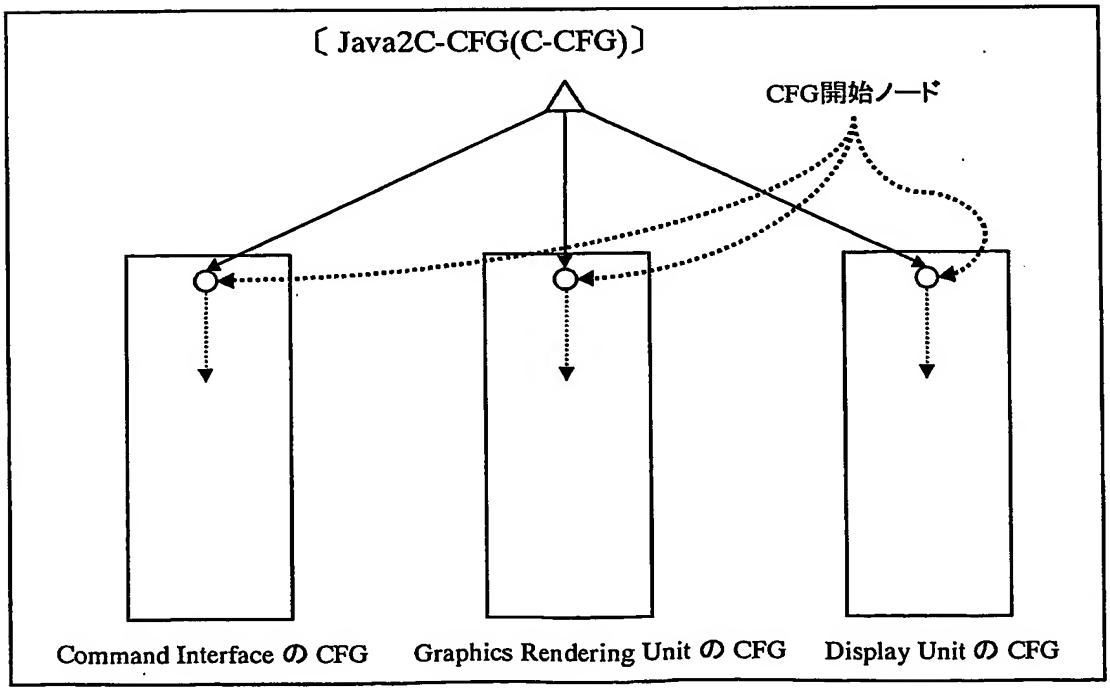
第35図


〔中間表現への変換〕
User Java code
Java2C-CFG
C-CFG
Abstraction of each CFG
C-CFG2C-TNFA
C-TNFA
C-TNFA2TNFA
TNFA
Abstraction of TNFA
Parametric Model Checker
Parameter Conditions
C-CFG2HDL
HDL
RPCTL

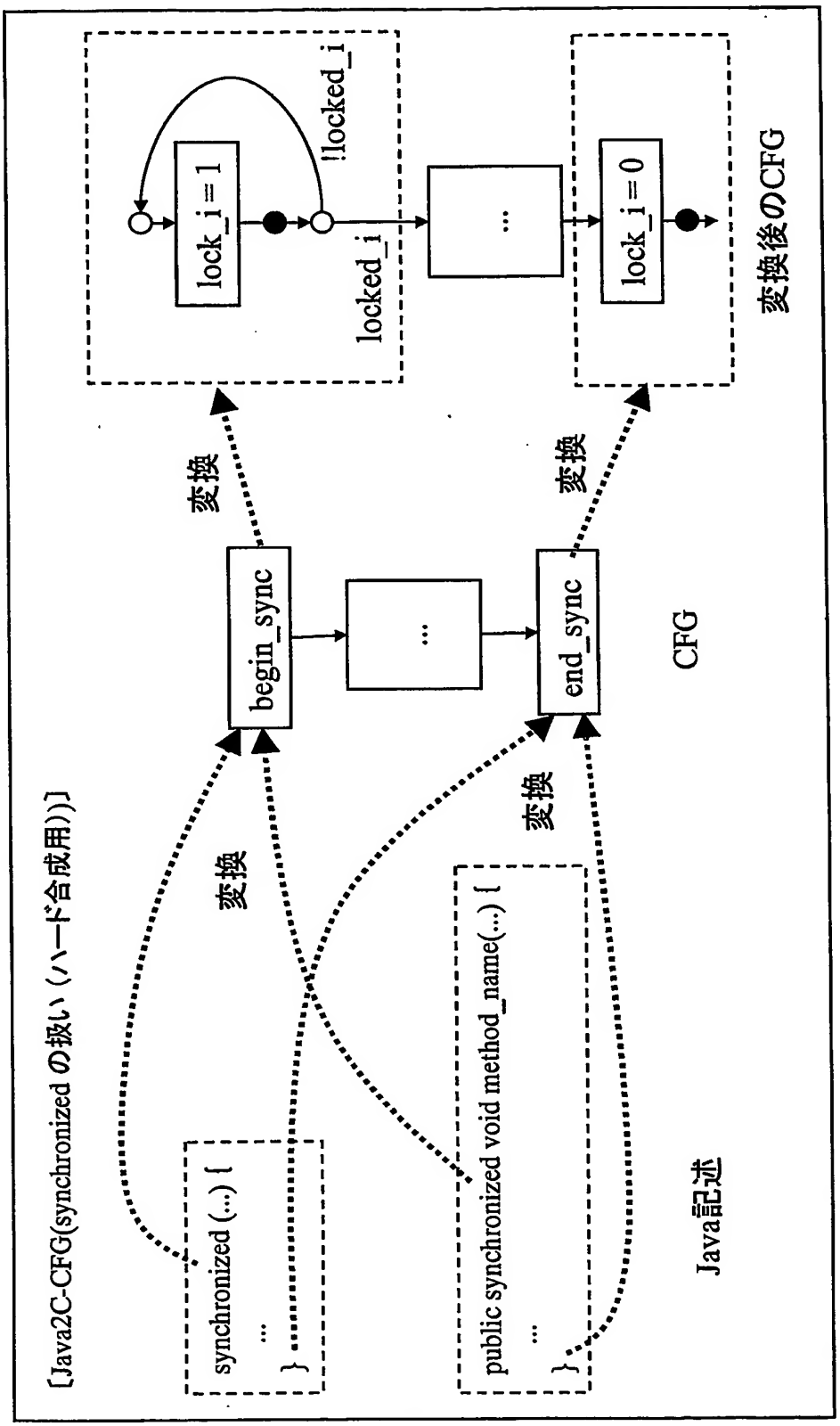
第３６図


〔Java2C-CFG(C-CFGの形式)〕
forkノード
Basic Blockノード
ループ開始ノード
分岐開始ノード
クロック境界ノード
分岐終了ノード
メソッド呼び出しノード
ループ終了ノード

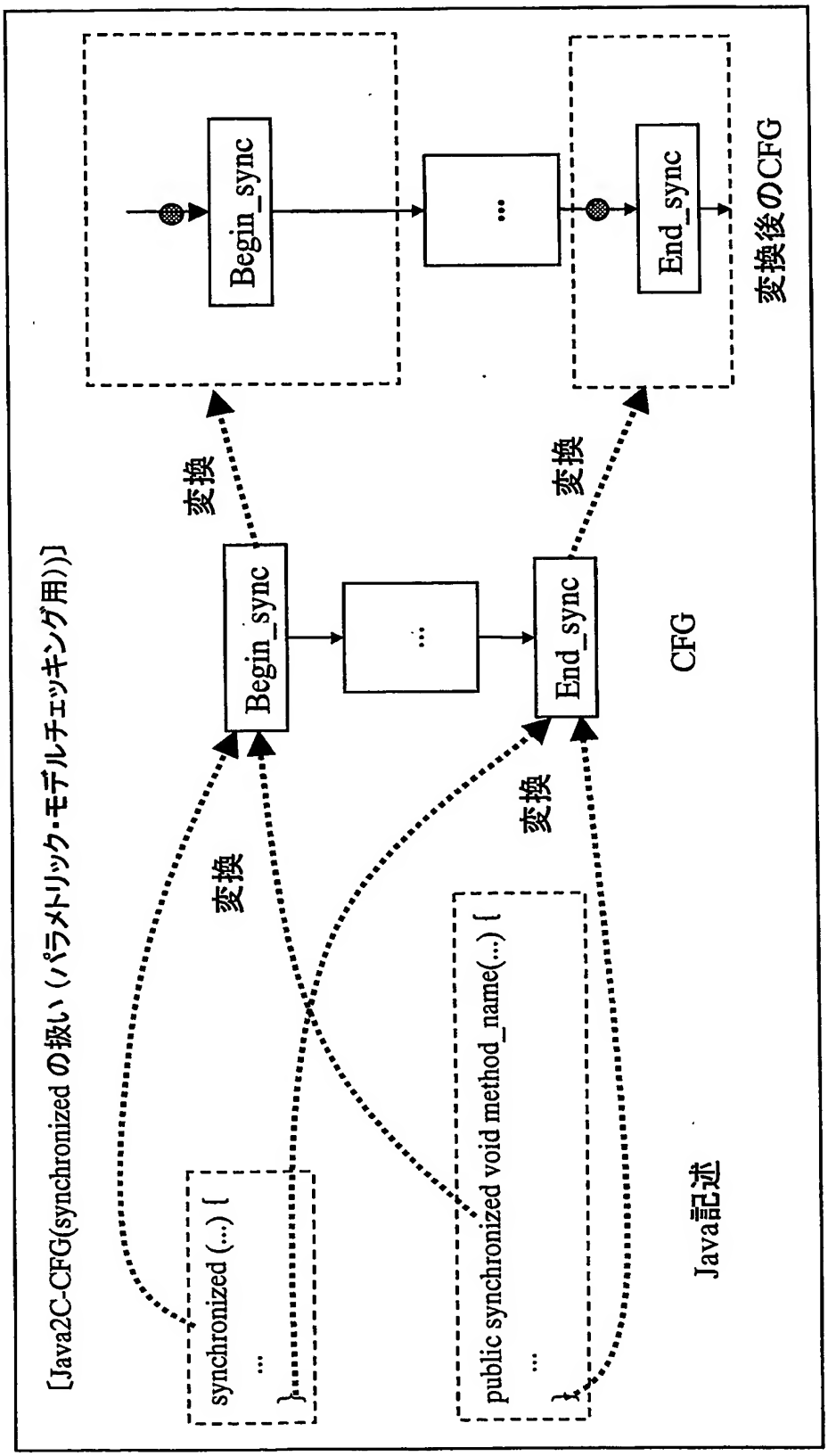
第４８図


〔Java2C-CFG(C-CFG)〕
CFG開始ノード
Command Interface の CFG
Graphics Rendering Unit の CFG
Display Unit の CFG

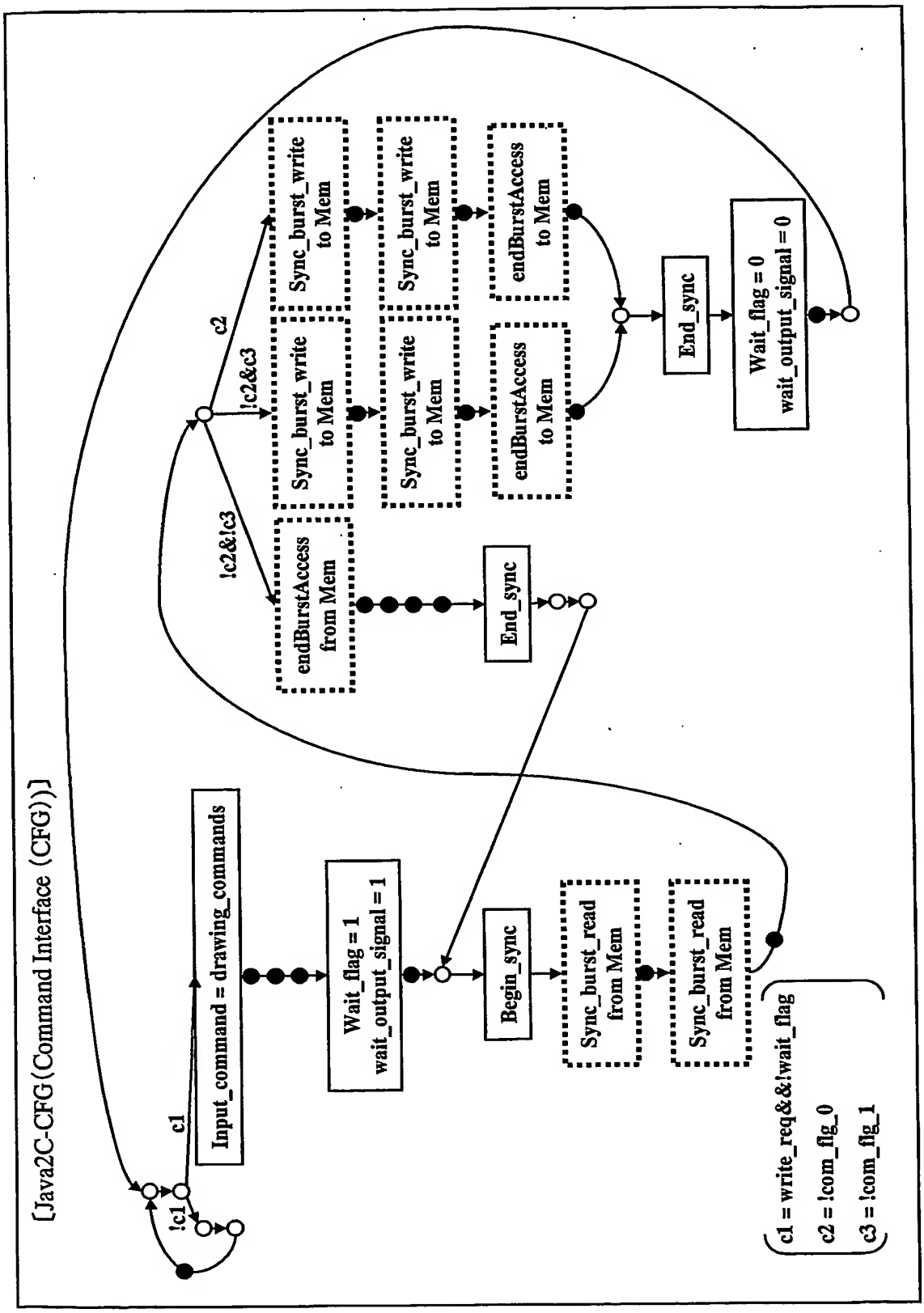
第３７図


〔Java2C-CFG(synchronized の扱い（ハード合成用））〕
synchronized (...) {
...
}
public synchronized void method_name(...) {
...
}
Java記述
変換
変換
変換
変換
begin_sync
...
end_sync
CFG
lock_i = 1
!locked_i
locked_i
...
lock_i = 0
変換後のCFG

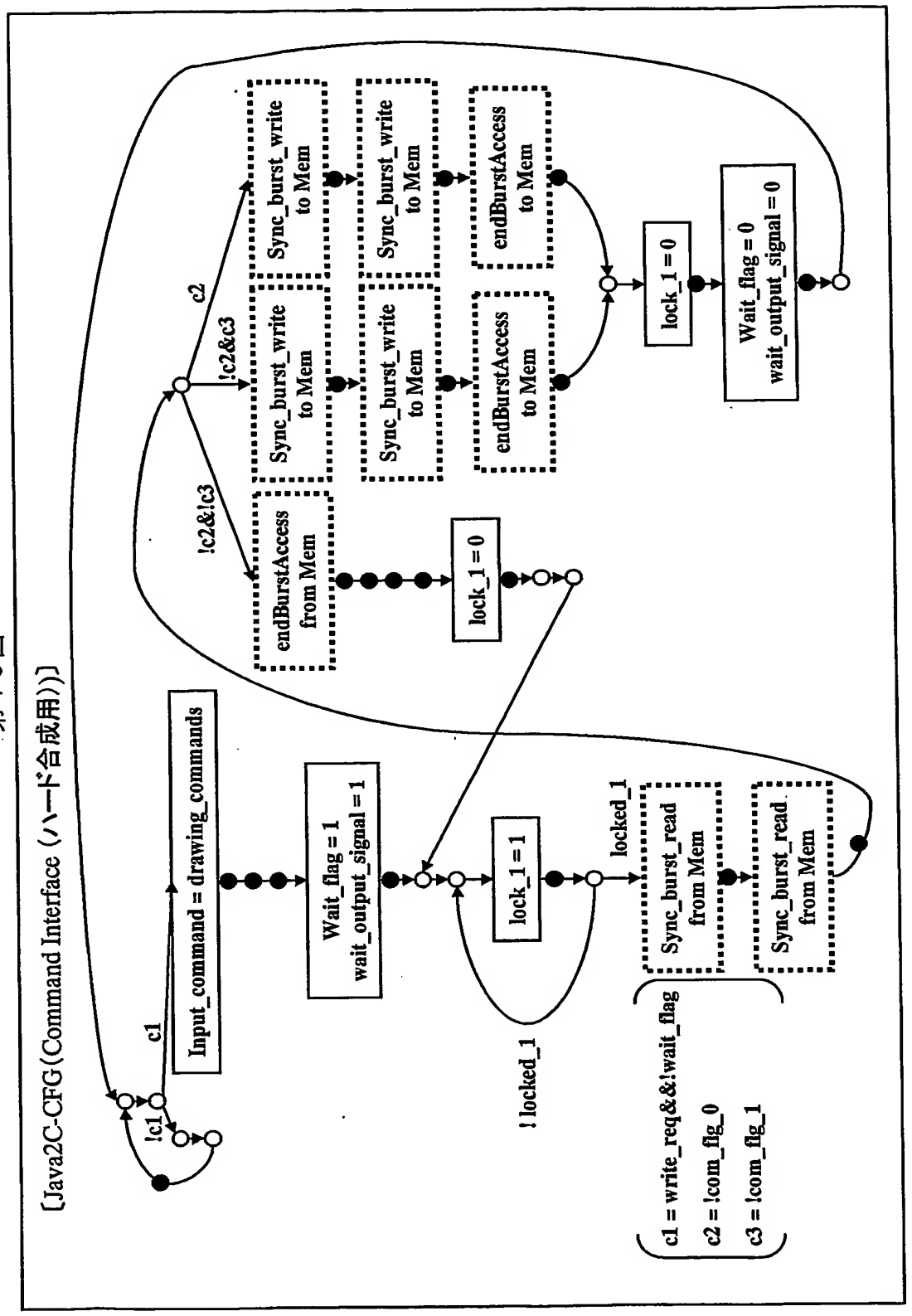
第３８図


〔Java2C-CFG(synchronized の扱い（パラメトリック・モデルチェッキング用））〕
Java記述
CFG
変換後のCFG
変換
変換
変換
変換
synchronized (...) {
...
}
public synchronized void method_name(...) {
...
}
Begin_sync
...
End_sync
Begin_sync
...
End_sync

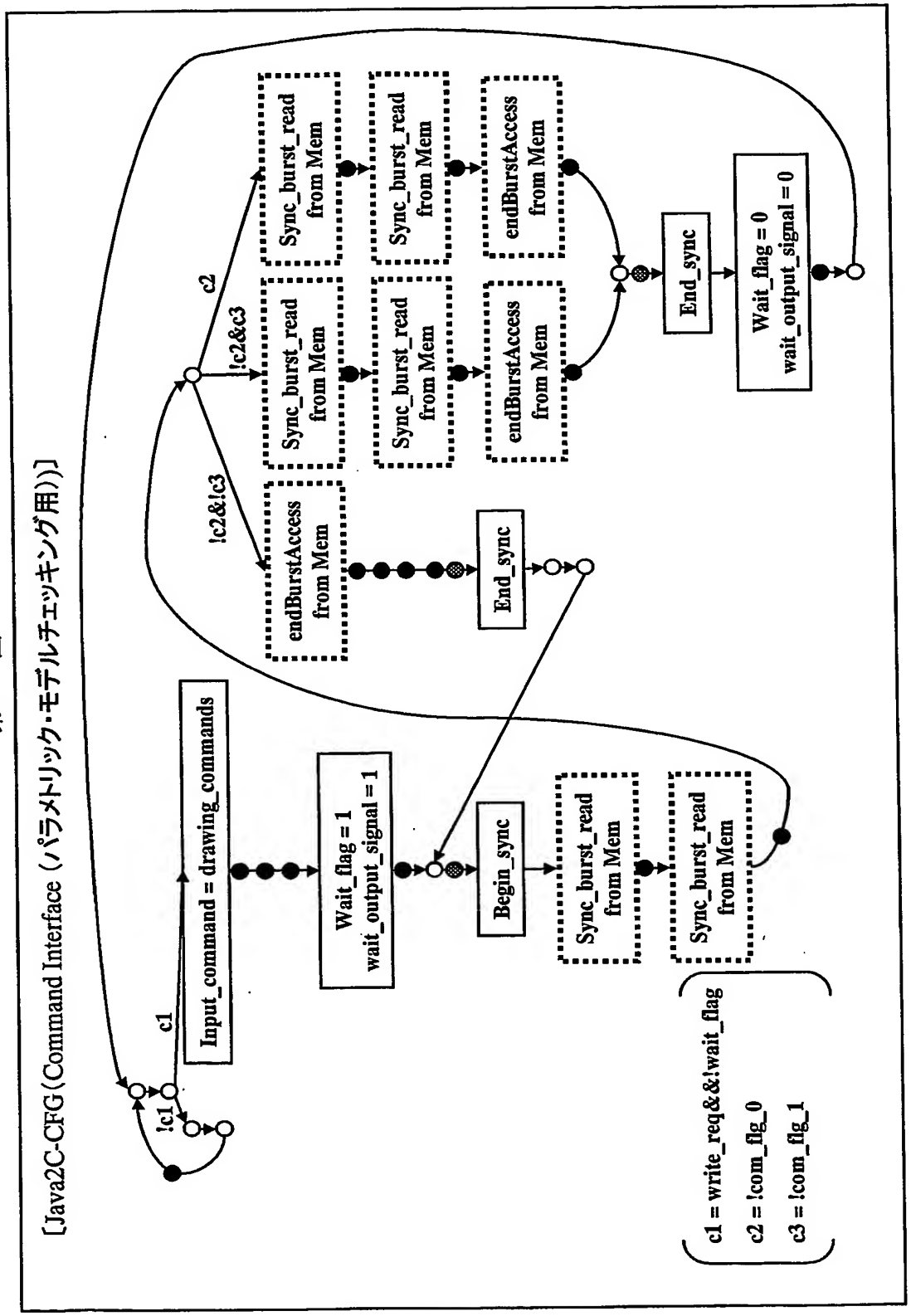
第39図


[Java2C-CFG(Command Interface (CFG))]
!c1
c1
Input_command = drawing_commands
Wait_flag = 1
wait_output_signal = 1
Begin_sync
Sync_burst_read from Mem
Sync_burst_read from Mem
c2
!c2&c3
!c2&!c3
Sync_burst_write to Mem
Sync_burst_write to Mem
endBurstAccess from Mem
Sync_burst_write to Mem
Sync_burst_write to Mem
endBurstAccess to Mem
End_sync
End_sync
Wait_flag = 0
wait_output_signal = 0
c1 = write_req&&!wait_flag
c2 = !com_flg_0
c3 = !com_flg_1

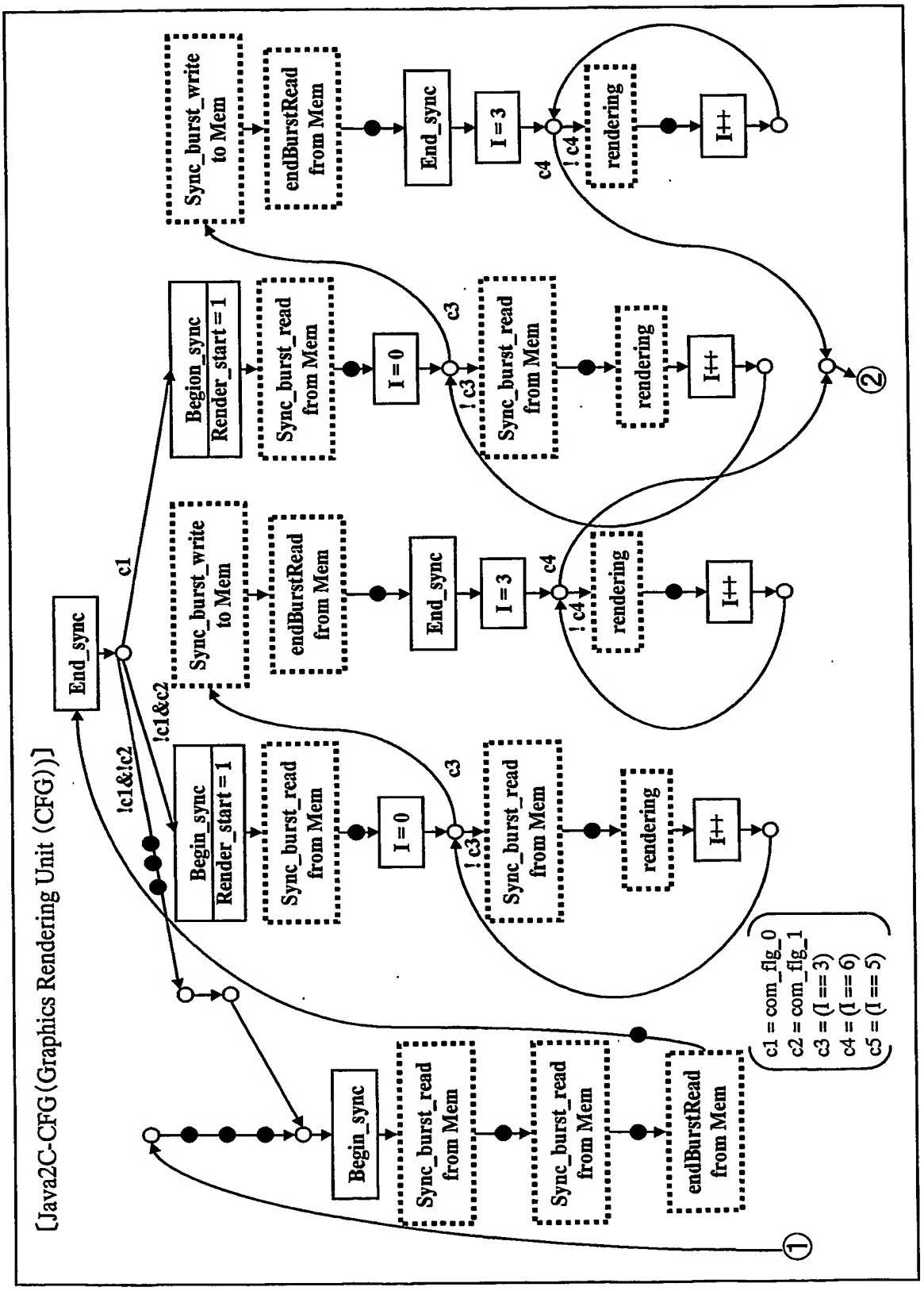
第４０図


〔Java2C-CFG(Command Interface (ハード合成用))〕
!c1
c1
Input_command = drawing_commands
Wait_flag = 1
wait_output_signal = 1
lock_1 = 1
locked_1
! locked_1
Sync_burst_read
from Mem
Sync_burst_read
from Mem
c1 = write_req&&!wait_flag
c2 = !com_flg_0
c3 = !com_flg_1
!c2&!c3
endBurstAccess
from Mem
lock_1 = 0
c2
!c2&c3
Sync_burst_write
to Mem
Sync_burst_write
to Mem
endBurstAccess
to Mem
Sync_burst_write
to Mem
Sync_burst_write
to Mem
endBurstAccess
to Mem
lock_1 = 0
Wait_flag = 0
wait_output_signal = 0

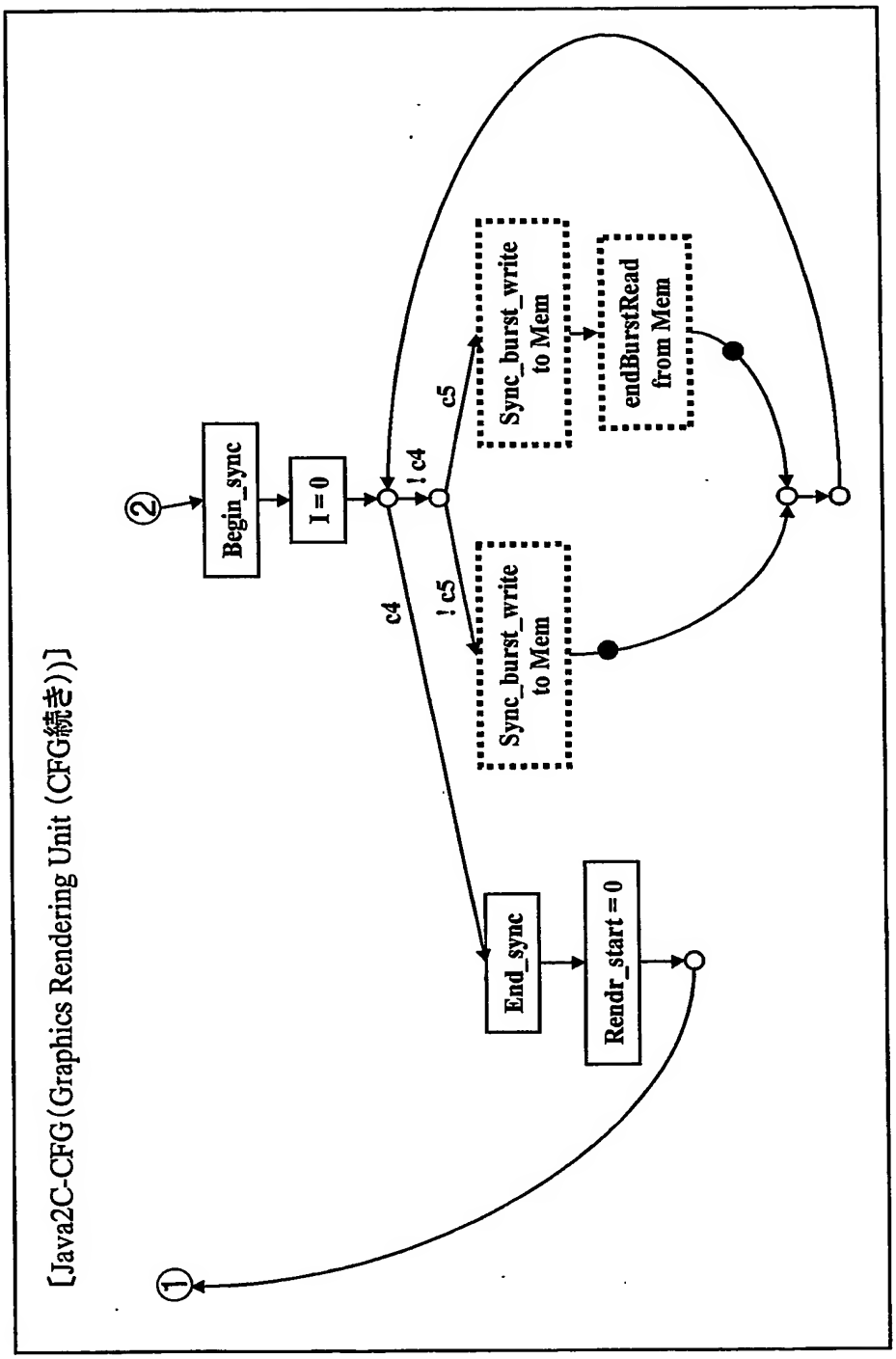
第４１図


[Java2C-CFG(Command Interface (パラメトリック・モデルチェッキング用))]
!c1
c1
Input_command = drawing_commands
Wait_flag = 1
wait_output_signal = 1
Begin_sync
Sync_burst_read from Mem
Sync_burst_read from Mem
c1 = write_req&&!wait_flag
c2 = !com_flg_0
c3 = !com_flg_1
c2
!c2&c3
!c2&!c3
Sync_burst_read from Mem
Sync_burst_read from Mem
Sync_burst_read from Mem
endBurstAccess from Mem
endBurstAccess from Mem
endBurstAccess from Mem
End_sync
End_sync
Wait_flag = 0
wait_output_signal = 0

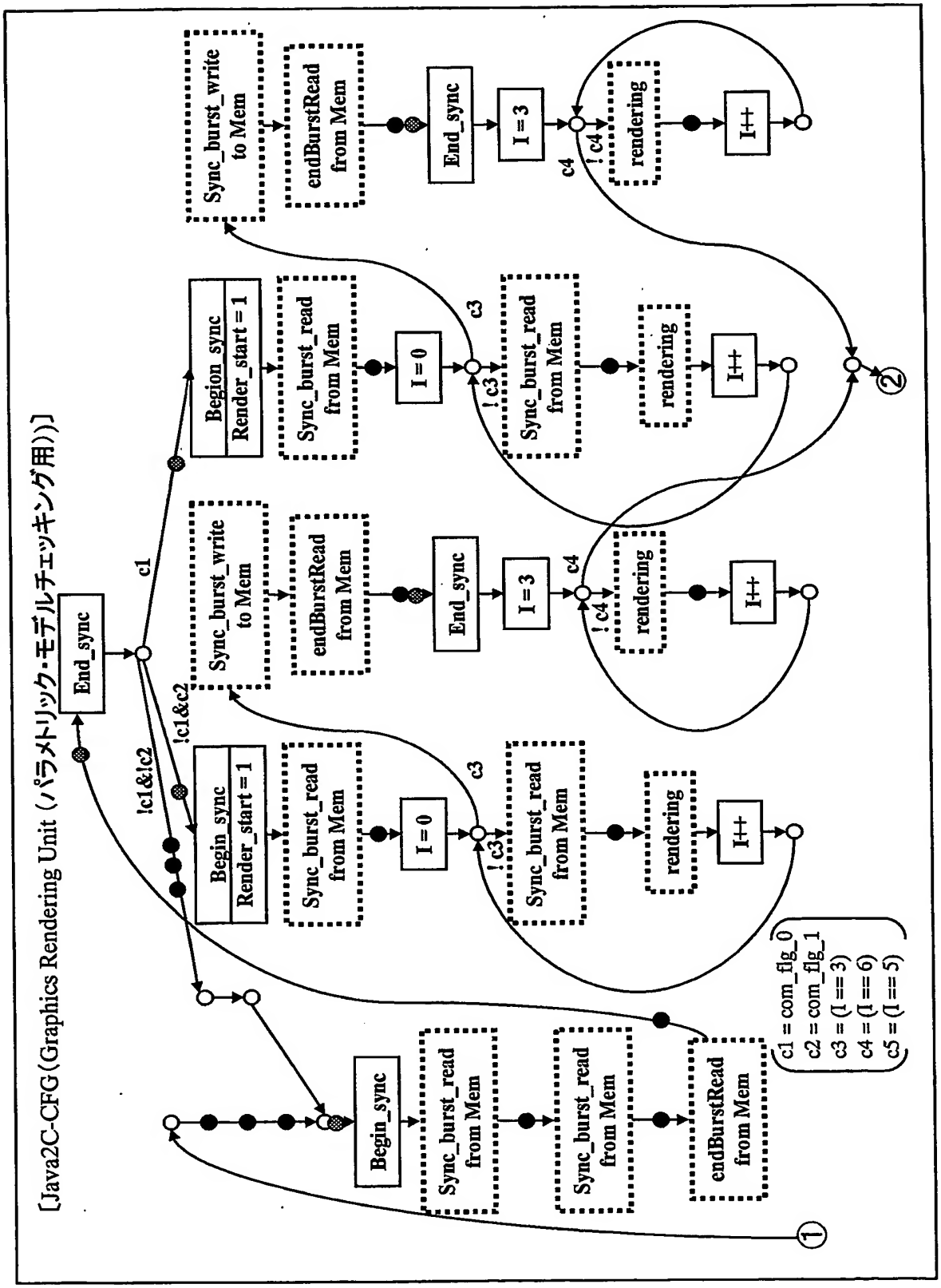
第42図


[Java2C-CFG(Graphics Rendering Unit (CFG))]
End_sync
c1
!c1&!c2
!c1&c2
Begin_sync
Render_start = 1
Sync_burst_read from Mem
I = 0
c3
! c3
Sync_burst_write to Mem
endBurstRead from Mem
End_sync
I = 3
c4
! c4
rendering
I++
Begin_sync
Sync_burst_read from Mem
Sync_burst_read from Mem
endBurstRead from Mem
c1 = com_flg_0
c2 = com_flg_1
c3 = (I == 3)
c4 = (I == 6)
c5 = (I == 5)
①
②

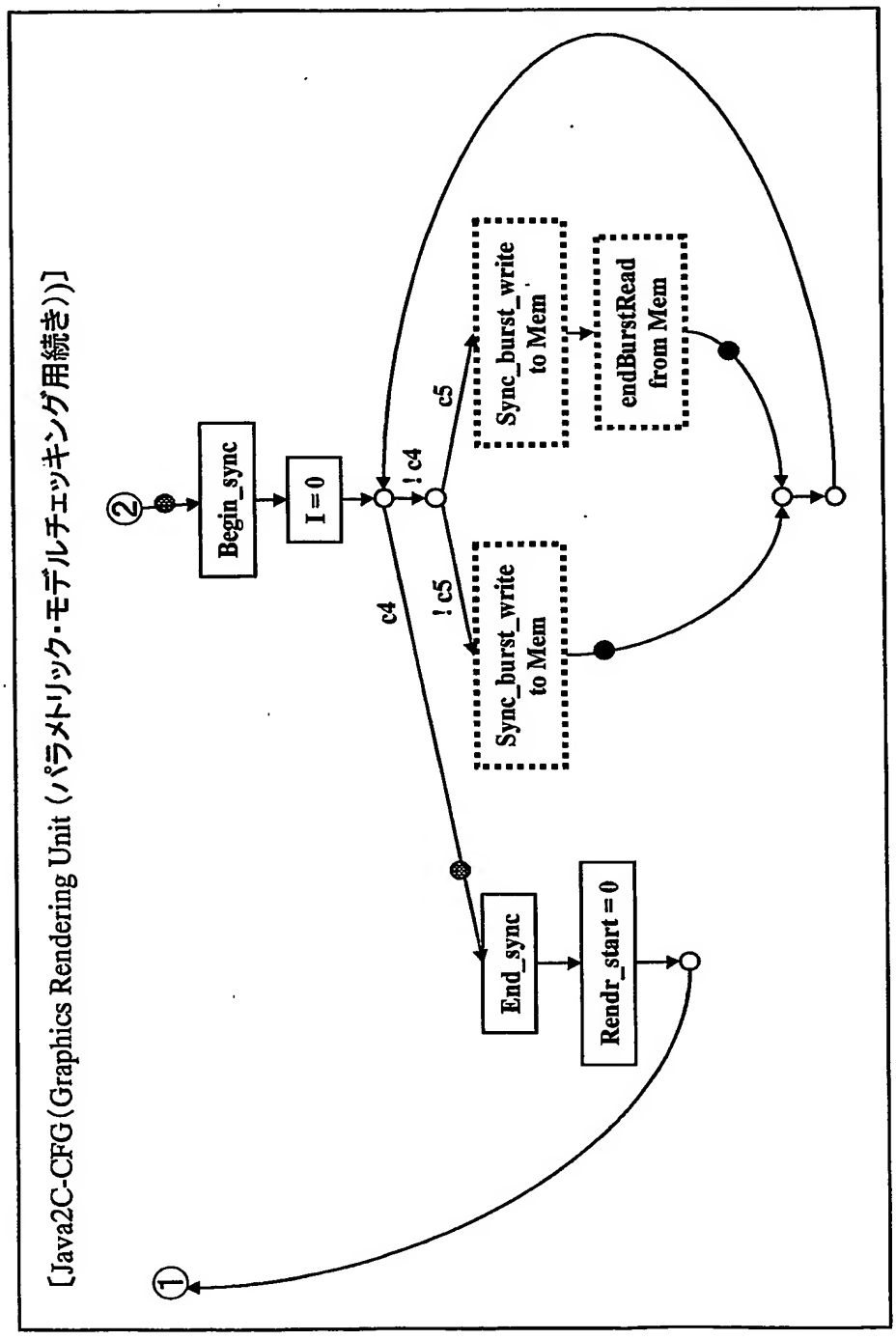
第４３図


[Java2C-CFG(Graphics Rendering Unit (CFG続き))]
②
Begin_sync
I = 0
! c4
c5
Sync_burst_write to Mem
endBurstRead from Mem
c4
! c5
Sync_burst_write to Mem
End_sync
Rendr_start = 0
①

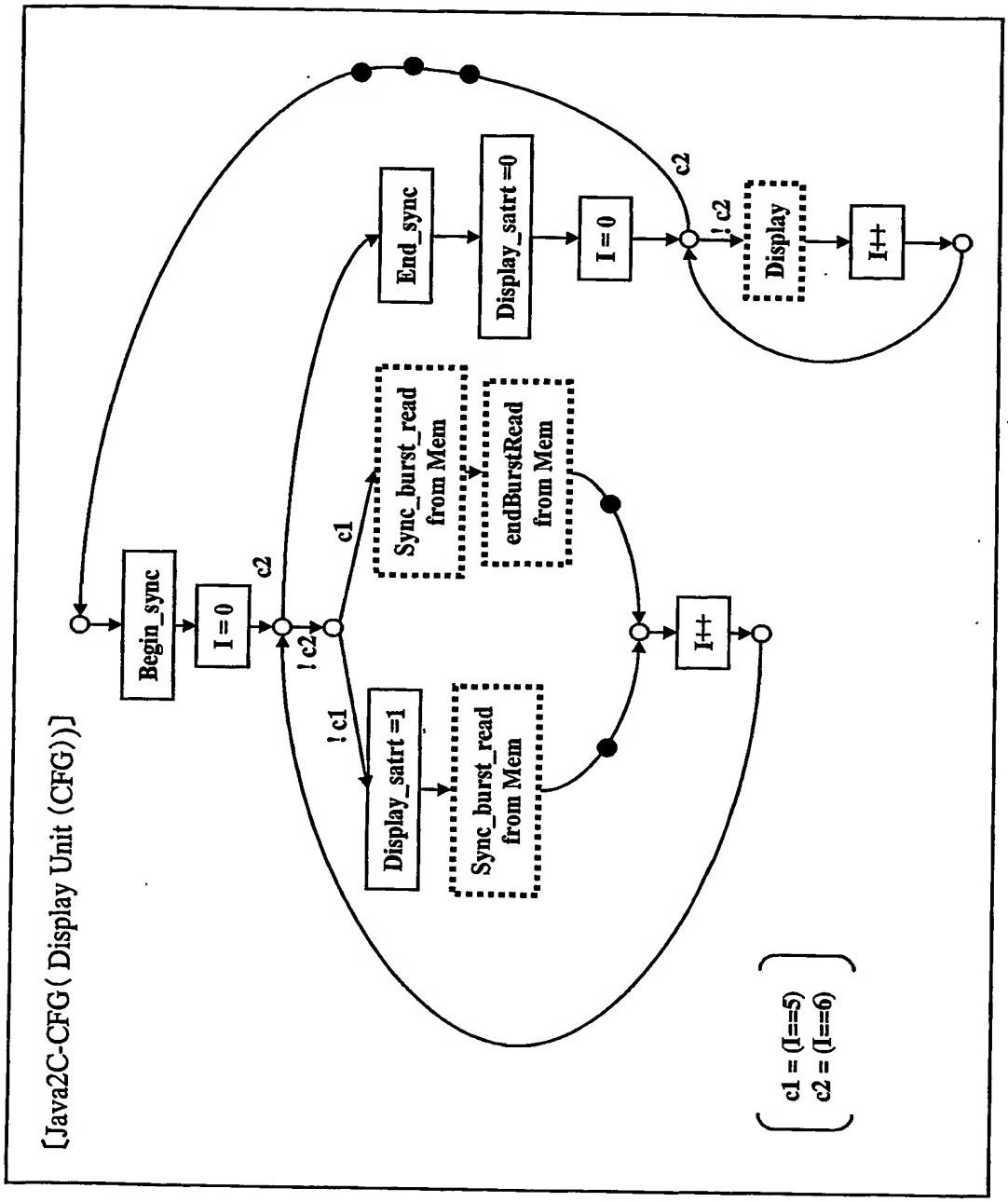
第４４図


[Java2C-CFG(Graphics Rendering Unit（パラメトリック・モデルチェッキング用))]
End_sync
c1
!c1&!c2
!c1&c2
Begin_sync
Render_start = 1
Sync_burst_read from Mem
I = 0
c3
! c3
Sync_burst_write to Mem
endBurstRead from Mem
End_sync
I = 3
c4
! c4
rendering
I++
Begin_sync
Sync_burst_read from Mem
Sync_burst_read from Mem
endBurstRead from Mem
c1 = com_flg_0
c2 = com_flg_1
c3 = (I == 3)
c4 = (I == 6)
c5 = (I == 5)
①
②

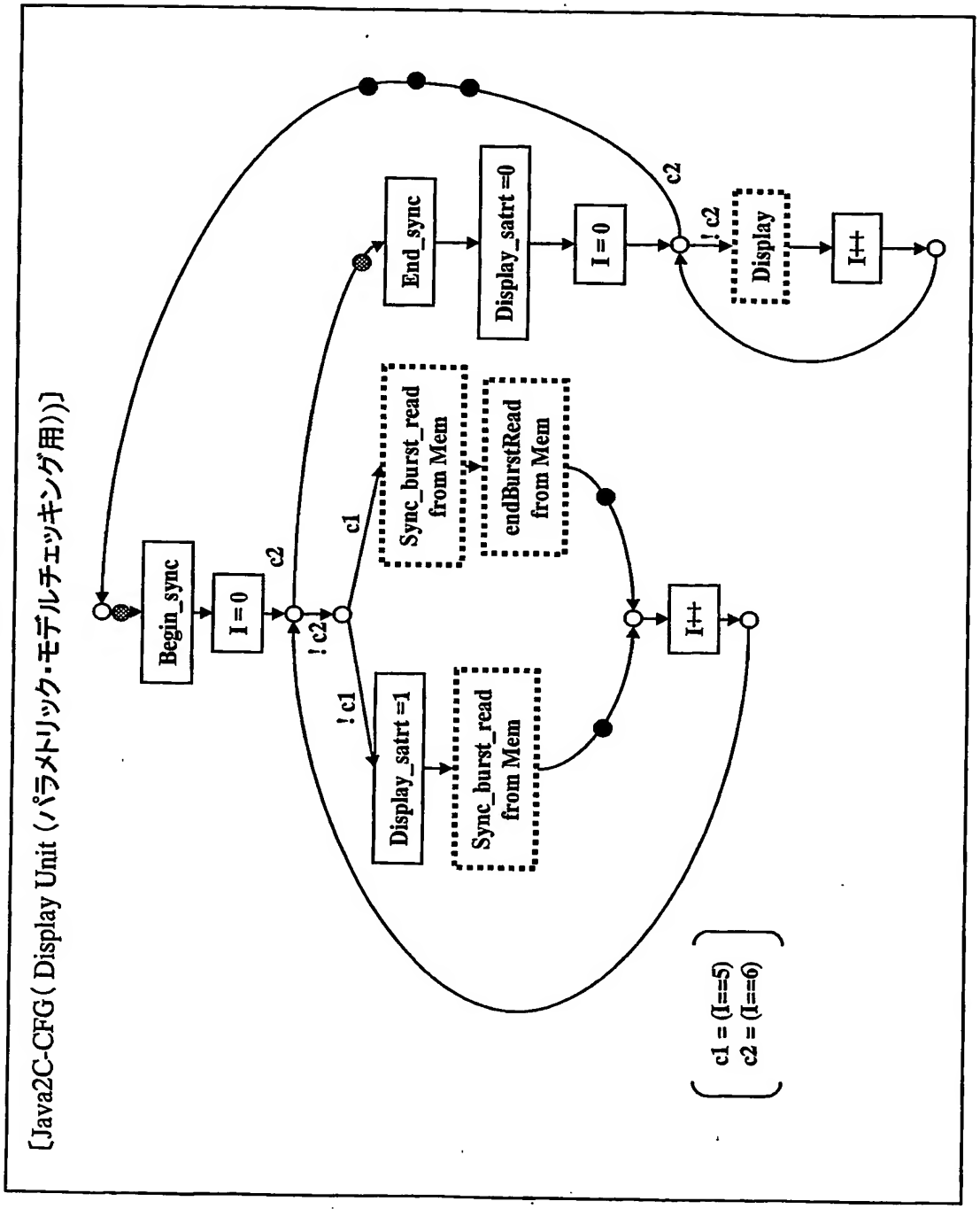
第４５図


〔Java2C-CFG(Graphics Rendering Unit (パラメトリック・モデルチェッキング用続き))〕
②
Begin_sync
I = 0
! c4
c5
Sync_burst_write to Mem
endBurstRead from Mem
c4
! c5
Sync_burst_write to Mem
End_sync
Rendr_start = 0
①

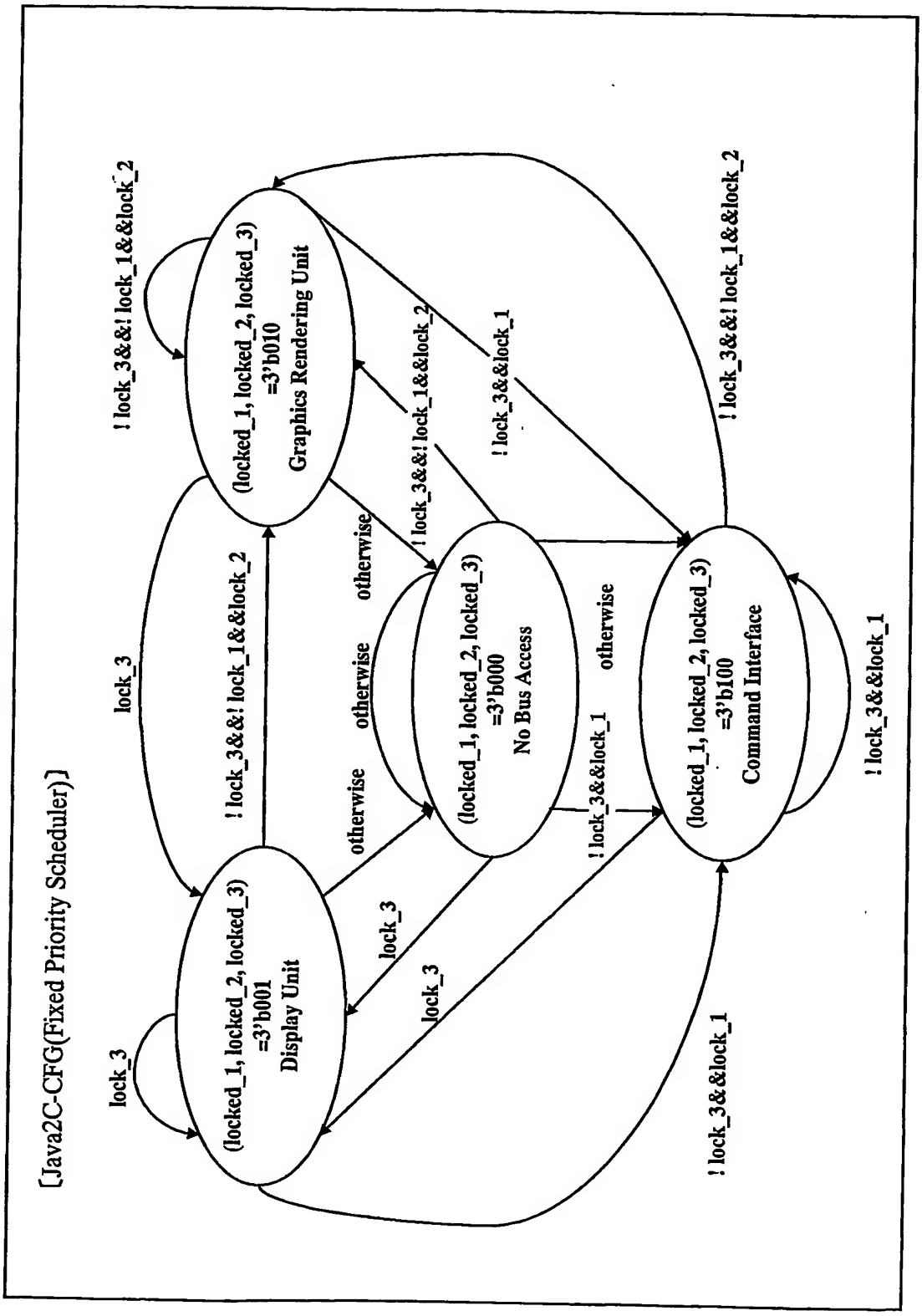
第46図


[Java2C-CFG(Display Unit (CFG))]
Begin_sync
I = 0
c2
! c2
c1
Sync_burst_read from Mem
endBurstRead from Mem
! c1
Display_satrt =1
Sync_burst_read from Mem
I++
End_sync
Display_satrt =0
I = 0
c2
! c2
Display
I++
c1 = (I==5)
c2 = (I==6)

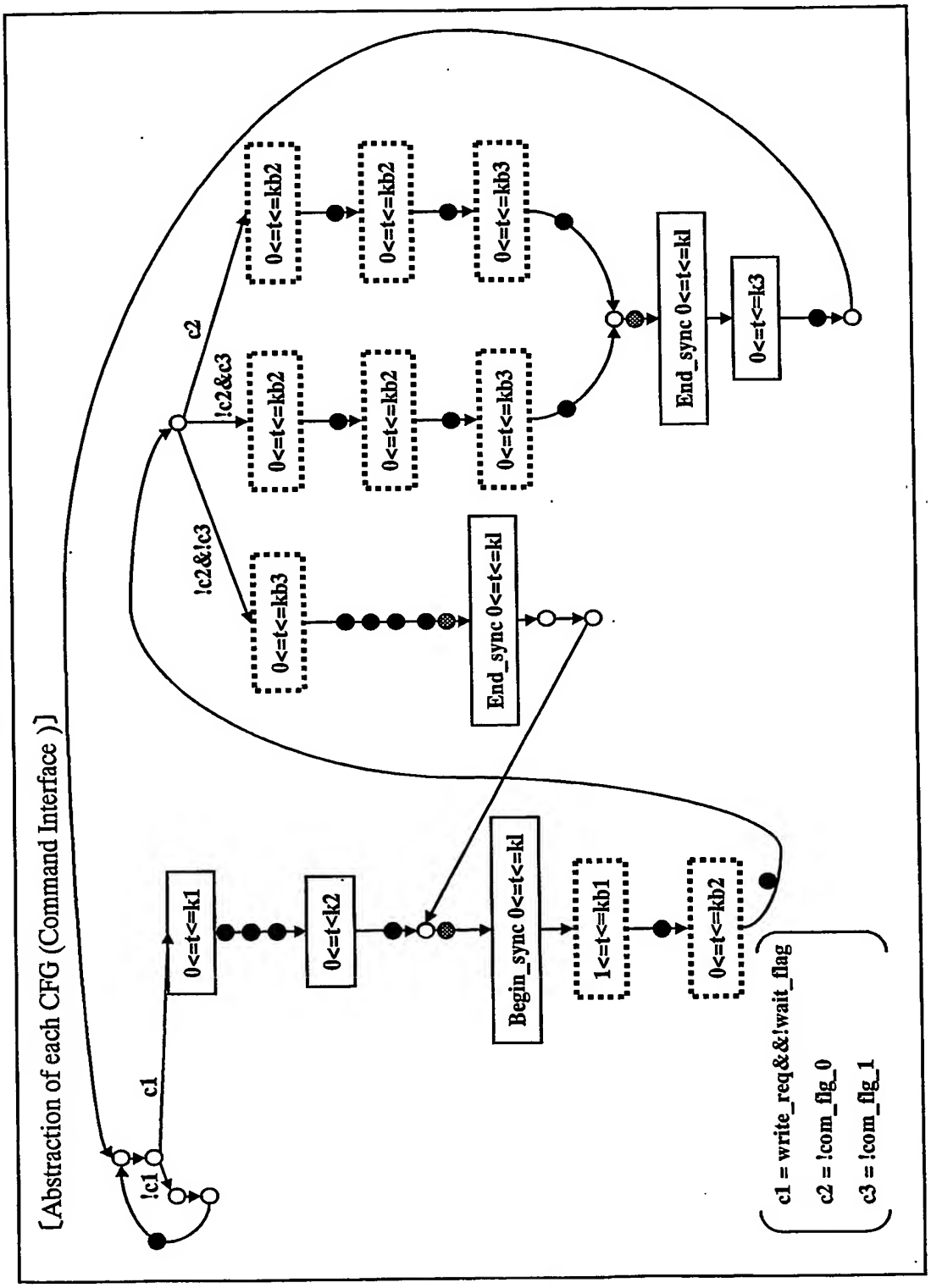
第４７図


〔Java2C-CFG(Display Unit (パラメトリック・モデルチェッキング用))〕
Begin_sync
I = 0
c2
! c2
c1
! c1
Sync_burst_read from Mem
endBurstRead from Mem
Display_satrt =1
Sync_burst_read from Mem
I++
End_sync
Display_satrt =0
I = 0
c2
! c2
Display
I++
c1 = (I==5)
c2 = (I==6)

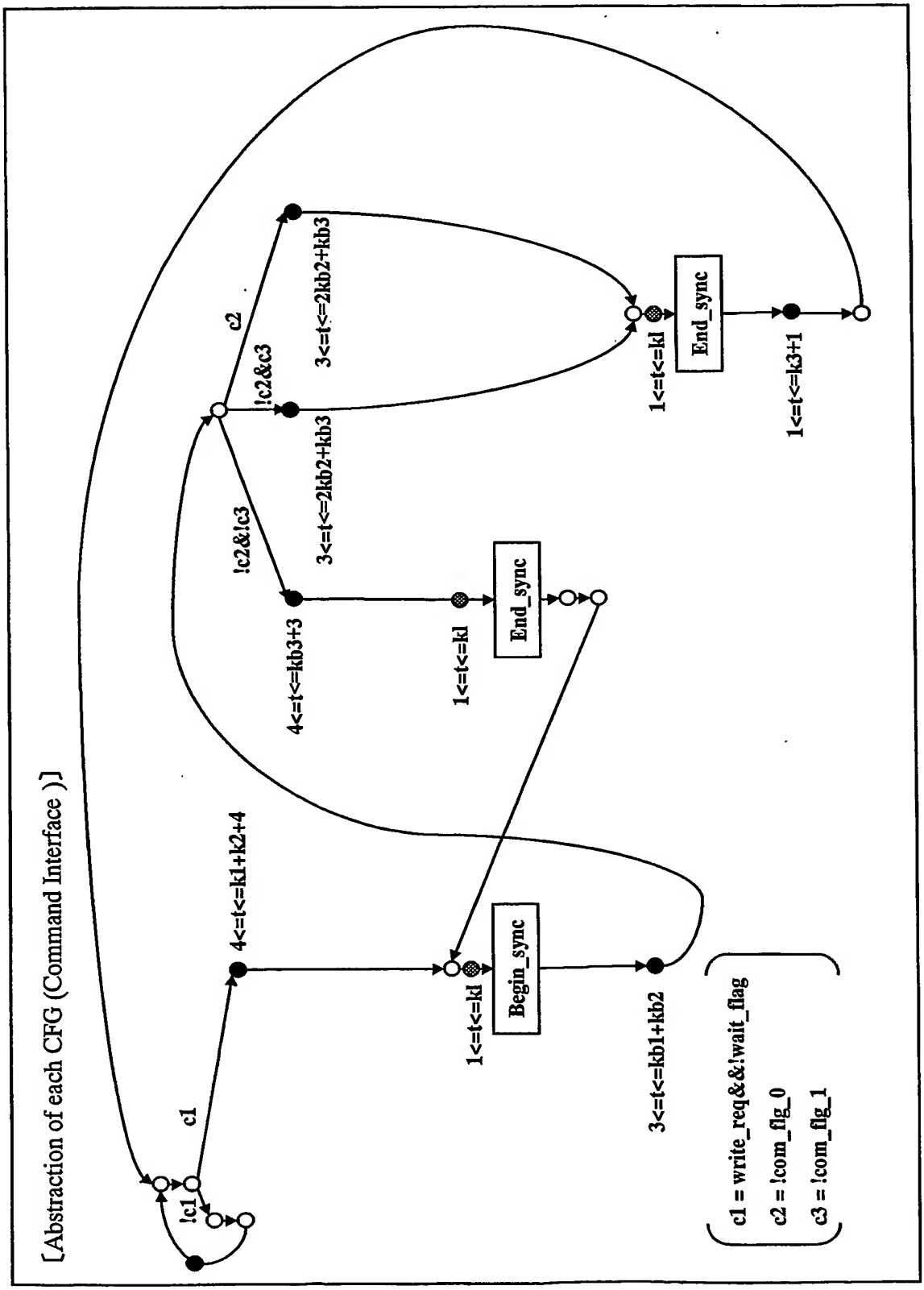
第４９図


[Java2C-CFG(Fixed Priority Scheduler)]
(locked_1, locked_2, locked_3)
=3'b001
Display Unit
(locked_1, locked_2, locked_3)
=3'b010
Graphics Rendering Unit
(locked_1, locked_2, locked_3)
=3'b000
No Bus Access
(locked_1, locked_2, locked_3)
=3'b100
Command Interface
lock_3
! lock_3&&! lock_1&&lock_2
! lock_3&&lock_1
otherwise
! lock_3&&lock_1

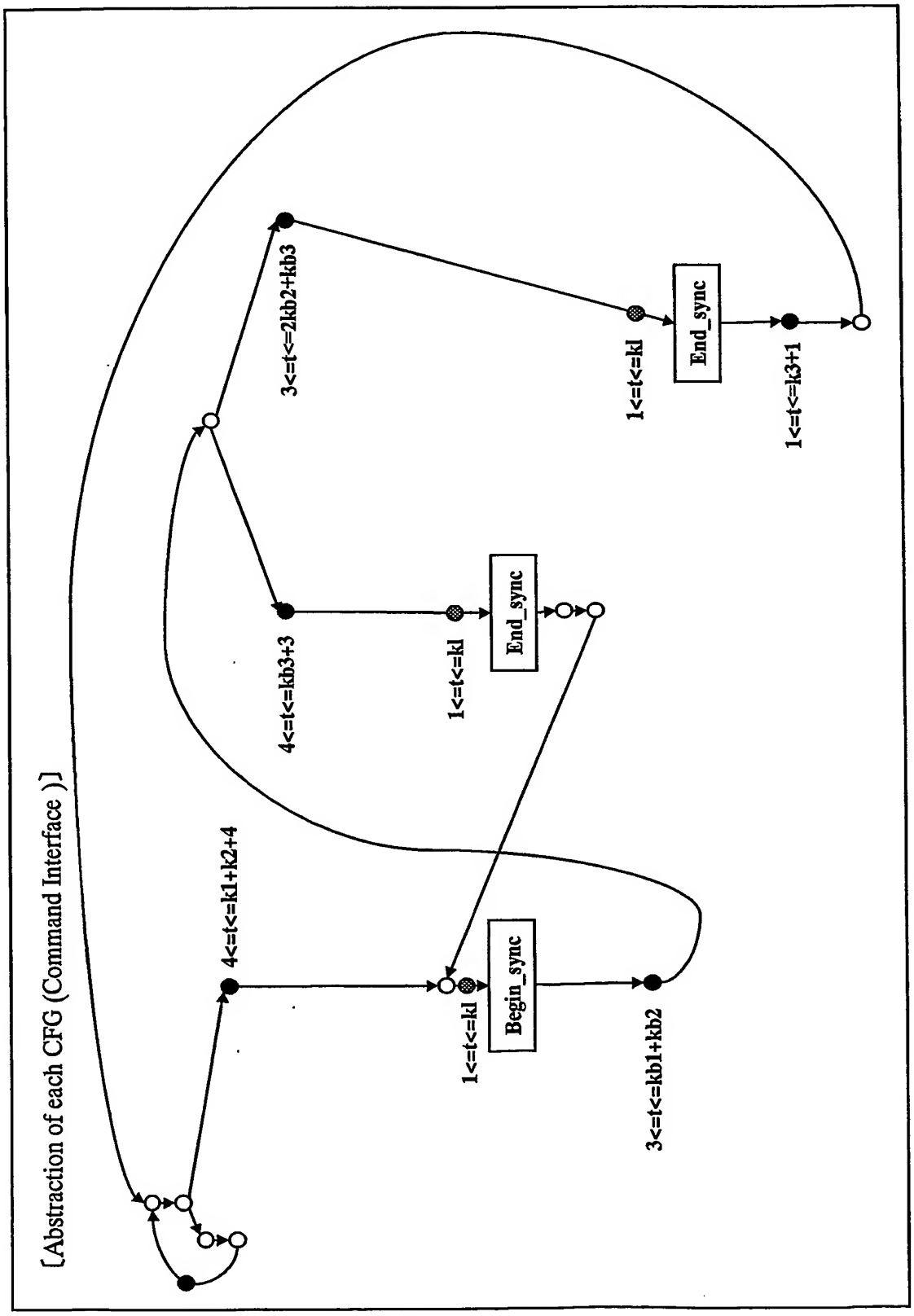
第５０図


[Abstraction of each CFG (Command Interface)]
!c1
c1
0<=t<=k1
0<t<k2
Begin_sync 0<=t<=k1
1<=t<=kb1
0<=t<=kb2
c2
!c2&c3
!c2&!c3
0<=t<=kb2
0<=t<=kb2
0<=t<=kb3
0<=t<=kb2
0<=t<=kb2
0<=t<=kb3
0<=t<=kb3
End_sync 0<=t<=k1
End_sync 0<=t<=k1
0<=t<=k3
c1 = write_req&&!wait_flag
c2 = !com_flg_0
c3 = !com_flg_1

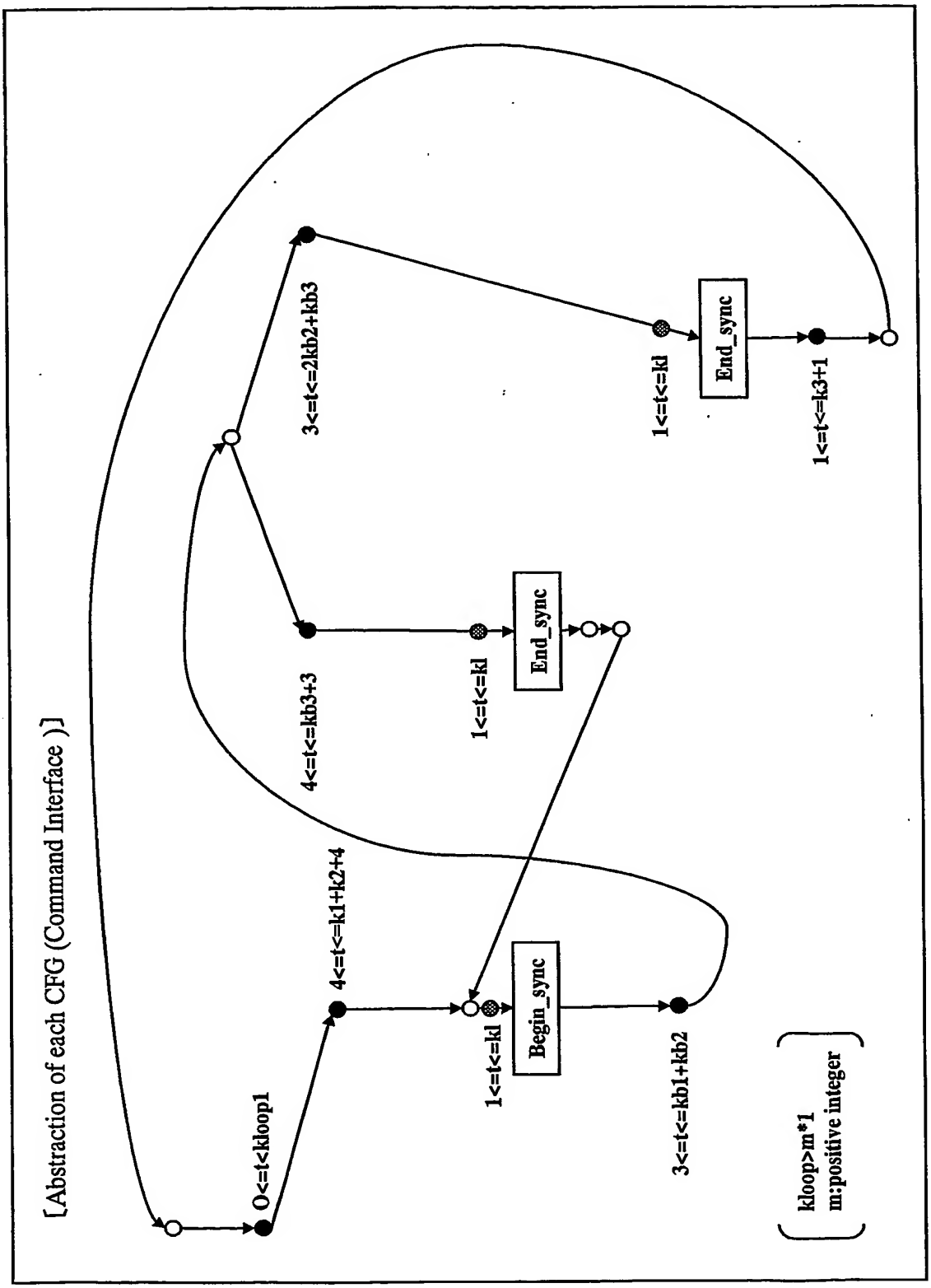
第５１図


[Abstraction of each CFG (Command Interface)]
!c1
c1
4<=t<=k1+k2+4
1<=t<=k1
Begin_sync
3<=t<=kb1+kb2
!c2&!c3
4<=t<=kb3+3
1<=t<=k1
End_sync
!c2&c3
3<=t<=2kb2+kb3
c2
3<=t<=2kb2+kb3
1<=t<=k1
End_sync
1<=t<=k3+1
c1 = write_req&&!wait_flag
c2 = !com_flg_0
c3 = !com_flg_1

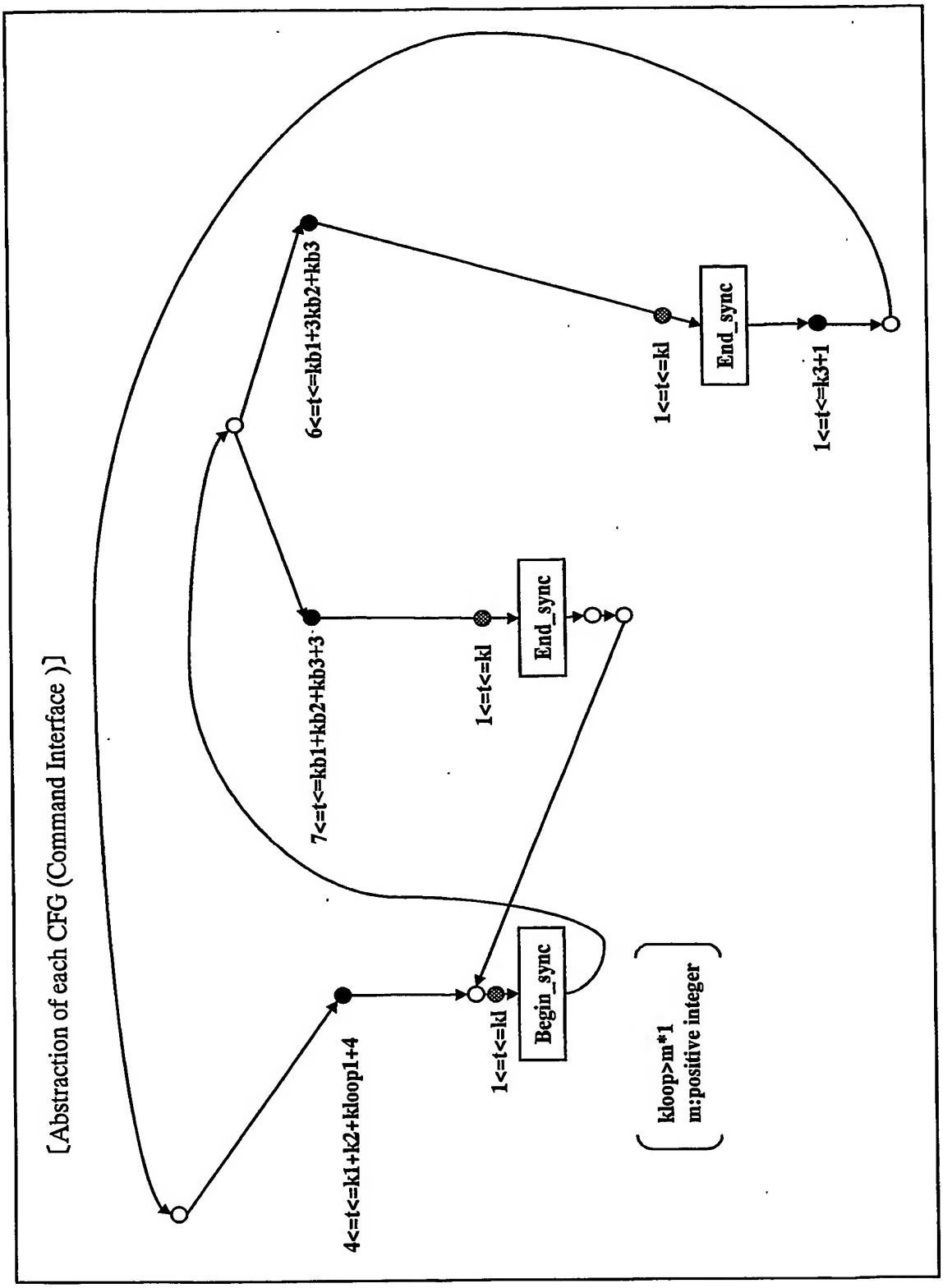
第５２図


[Abstraction of each CFG (Command Interface)]
4<=t<=k1+k2+4
1<=t<=k1
Begin_sync
3<=t<=kb1+kb2
4<=t<=kb3+3
1<=t<=k1
End_sync
3<=t<=2kb2+kb3
1<=t<=k1
End_sync
1<=t<=k3+1

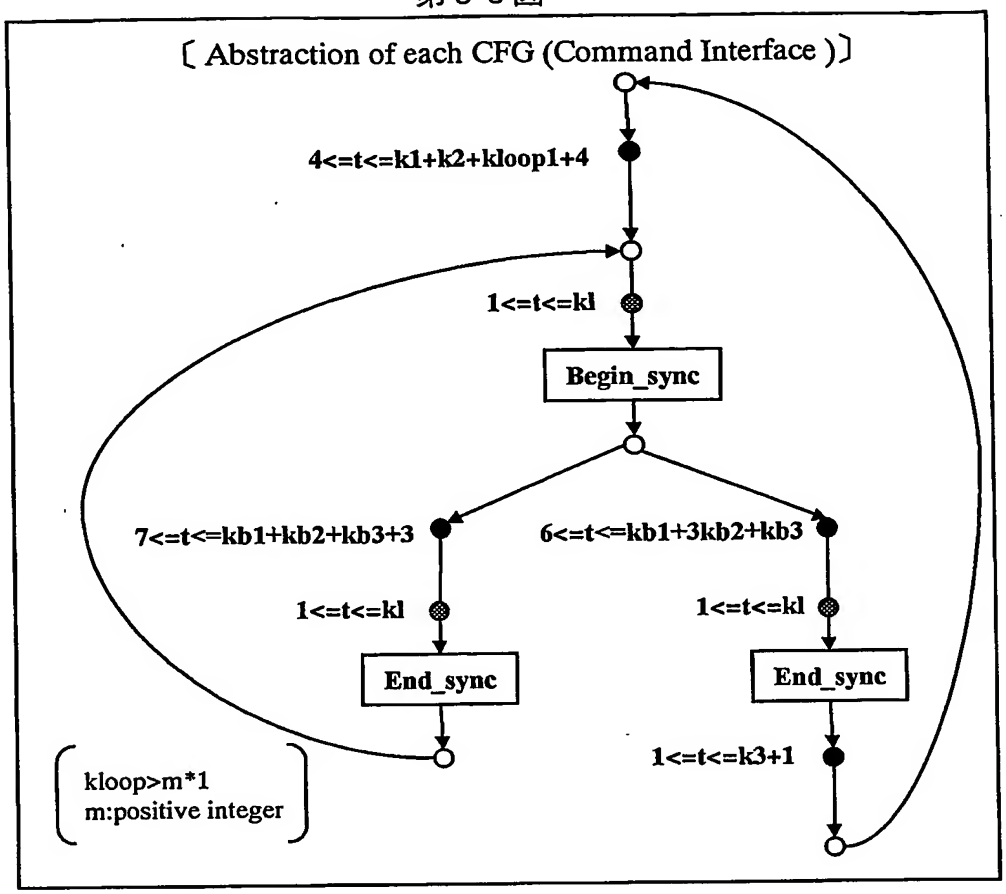
第５３図


[Abstraction of each CFG (Command Interface)]
0<=t<kloop1
4<=t<=k1+k2+4
1<=t<=k1
Begin_sync
3<=t<=kb1+kb2
4<=t<=kb3+3
1<=t<=k1
End_sync
3<=t<=2kb2+kb3
1<=t<=k1
End_sync
1<=t<=k3+1
kloop>m*1
m:positive integer

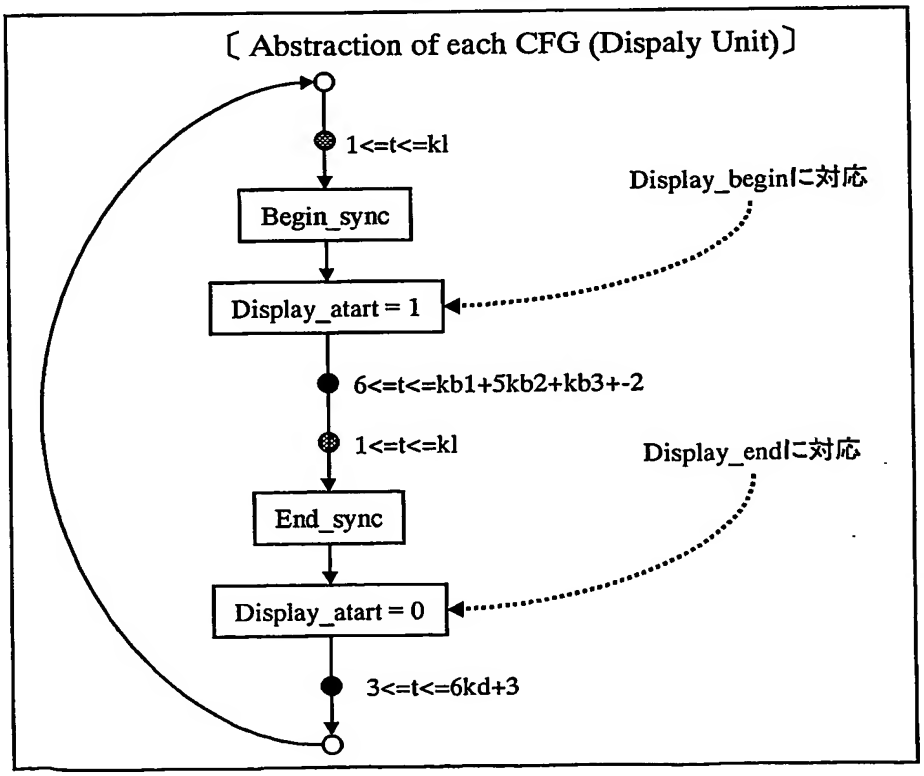
第５４図


〔Abstraction of each CFG (Command Interface)〕
4<=t<=k1+k2+kloop1+4
1<=t<=kl
Begin_sync
kloop>m*1
m:positive integer
7<=t<=kb1+kb2+kb3+3
1<=t<=kl
End_sync
6<=t<=kb1+3kb2+kb3
1<=t<=kl
End_sync
1<=t<=k3+1

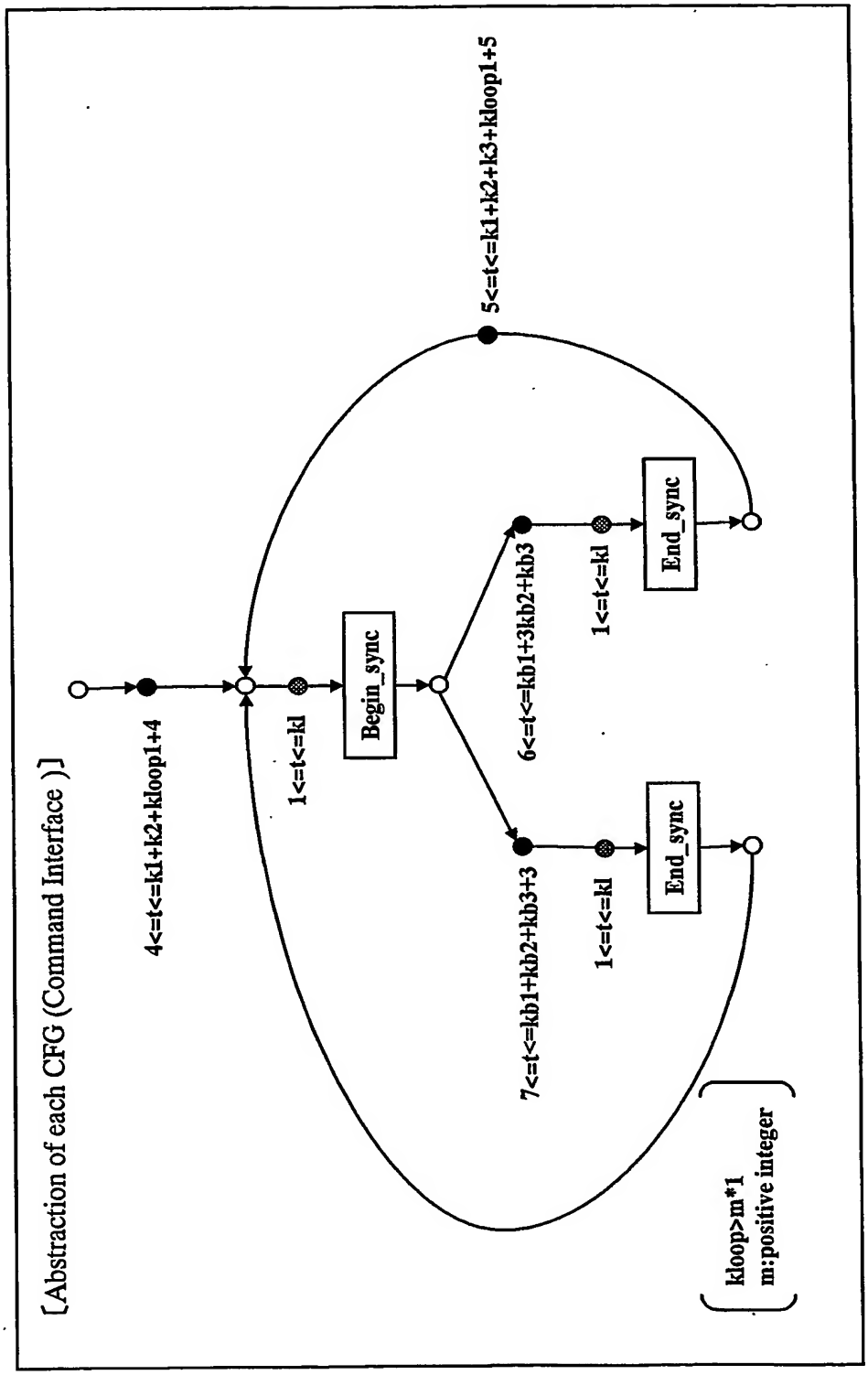
第５５図


〔Abstraction of each CFG (Command Interface)〕
4<=t<=k1+k2+kloop1+4
1<=t<=kl
Begin_sync
7<=t<=kb1+kb2+kb3+3
6<=t<=kb1+3kb2+kb3
1<=t<=kl
1<=t<=kl
End_sync
End_sync
1<=t<=k3+1
kloop>m*1
m:positive integer

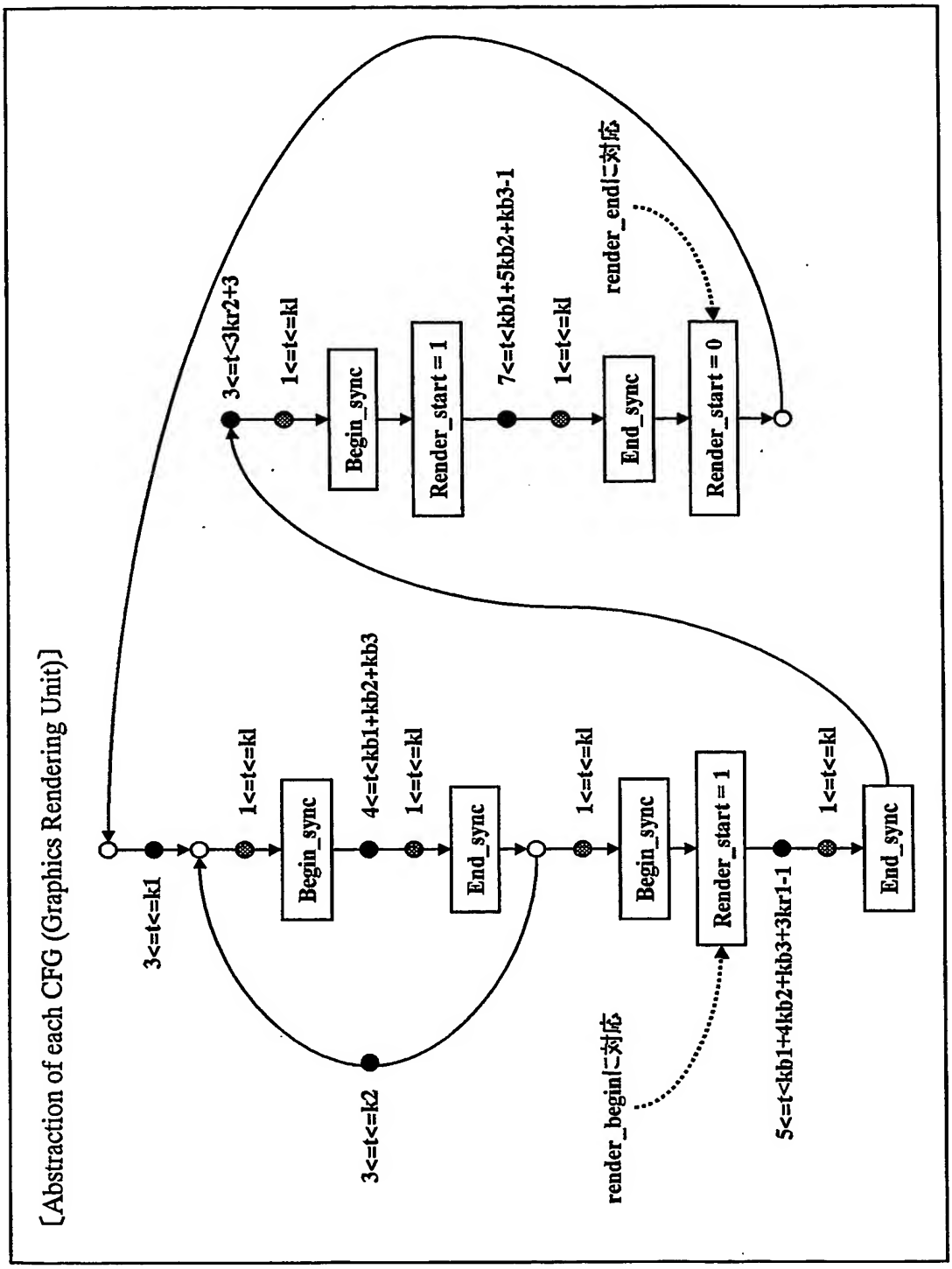
第５８図


〔Abstraction of each CFG (Dispaly Unit)〕
1<=t<=kl
Display_beginに対応
Begin_sync
Display_atart = 1
6<=t<=kb1+5kb2+kb3+-2
1<=t<=kl
Display_endに対応
End_sync
Display_atart = 0
3<=t<=6kd+3

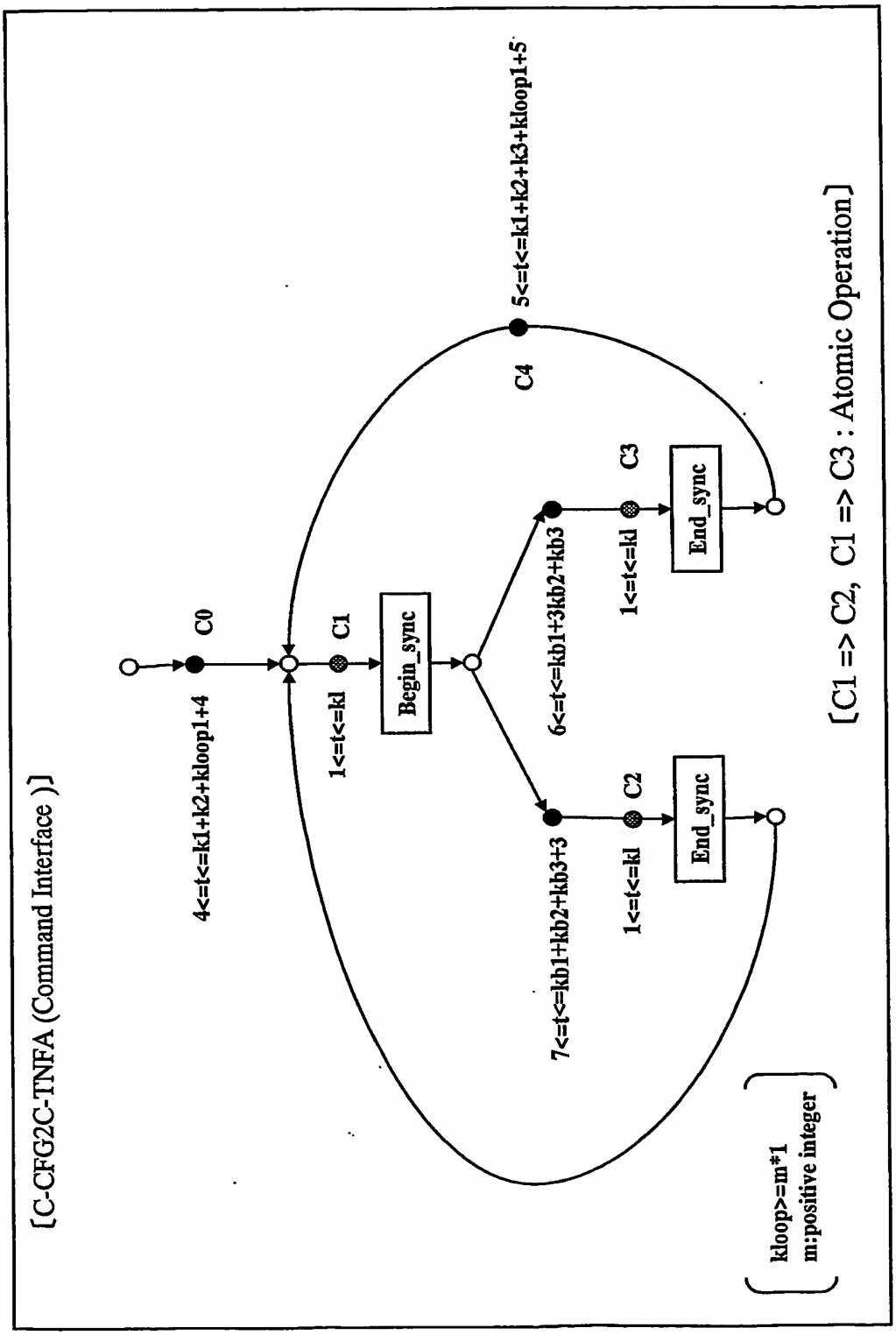
第56図


[Abstraction of each CFG (Command Interface)]
4<=t<=k1+k2+kloop1+4
1<=t<=k1
Begin_sync
5<=t<=k1+k2+k3+kloop1+5
6<=t<=kb1+3kb2+kb3
1<=t<=k1
End_sync
7<=t<=kb1+kb2+kb3+3
1<=t<=k1
End_sync
kloop>m*1
m:positive integer

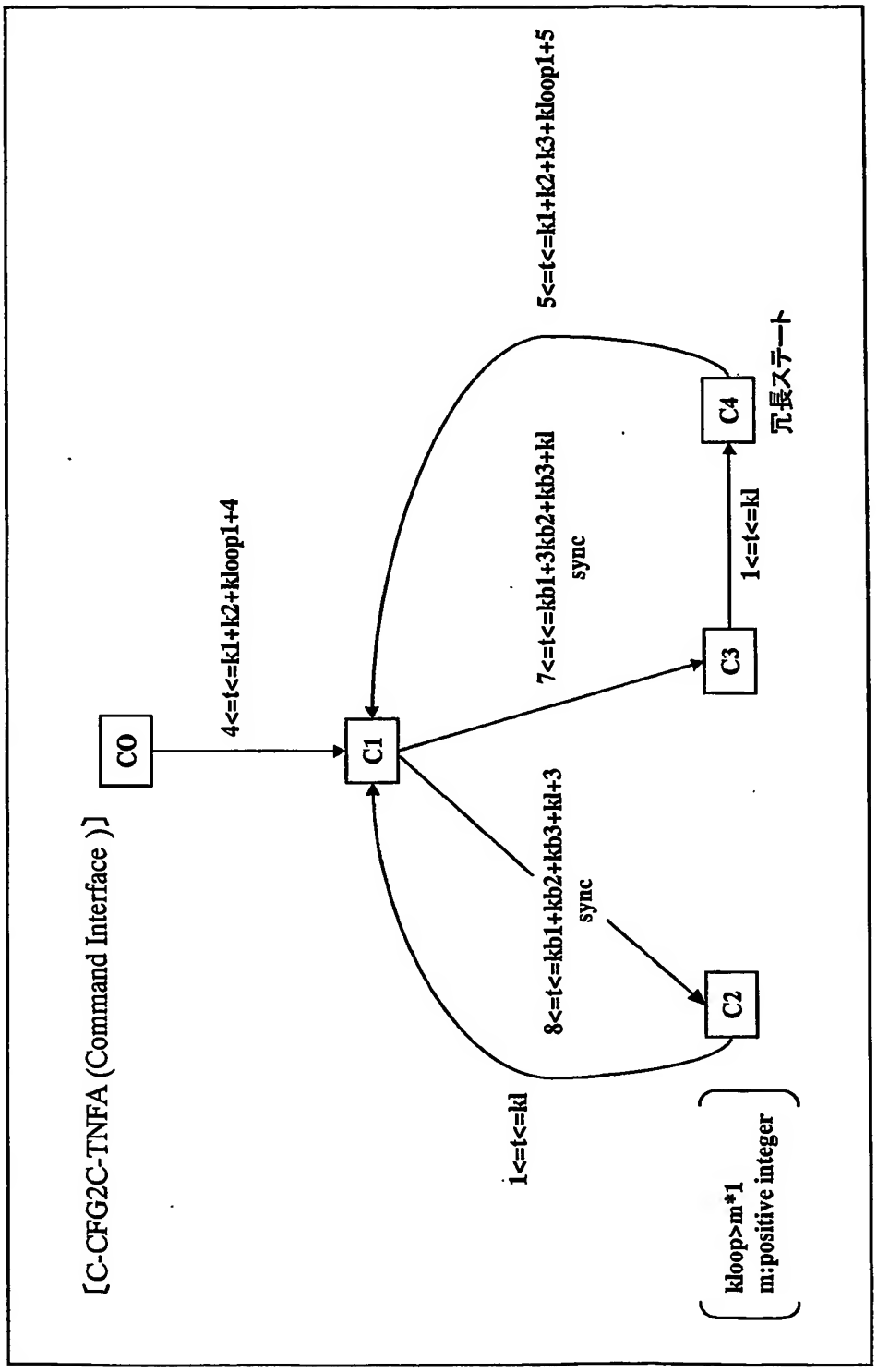
第57図


[Abstraction of each CFG (Graphics Rendering Unit)]
3<=t<=k1
1<=t<=kl
Begin_sync
4<=t<kb1+kb2+kb3
1<=t<=kl
End_sync
3<=t<=k2
1<=t<=kl
Begin_sync
render_begin1に対応
Render_start = 1
5<=t<kb1+4kb2+kb3+3kr1-1
1<=t<=kl
End_sync
3<=t<3kr2+3
1<=t<=kl
Begin_sync
Render_start = 1
7<=t<kb1+5kb2+kb3-1
1<=t<=kl
End_sync
render_end1に対応
Render_start = 0

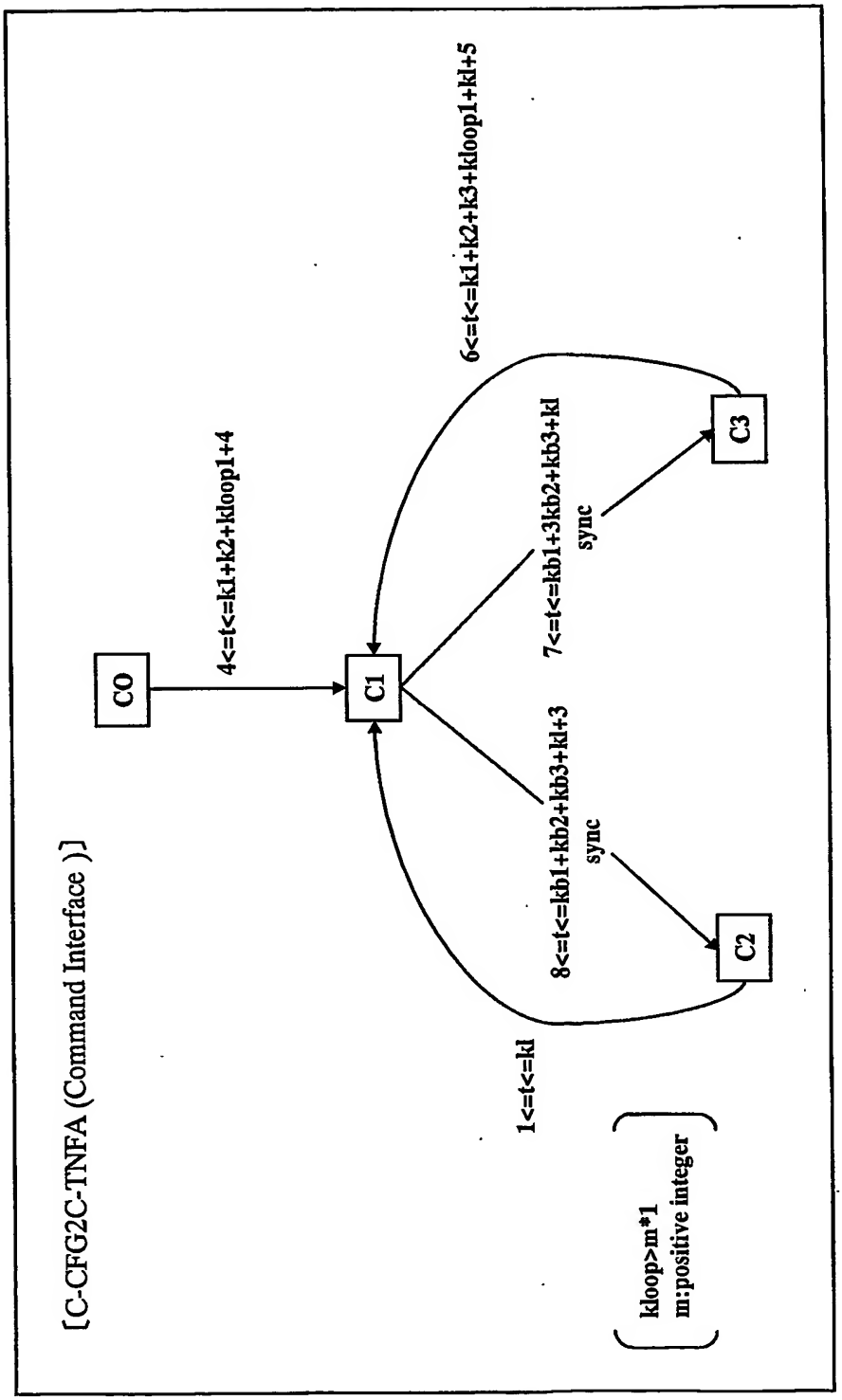
第59図


[C-CFG2C-TNFA (Command Interface)]
C0
4<=t<=k1+k2+kloop1+4
C1
1<=t<=k1
Begin_sync
6<=t<=kb1+3kb2+kb3
C3
1<=t<=k1
End_sync
C4
5<=t<=k1+k2+k3+kloop1+5
7<=t<=kb1+kb2+kb3+3
C2
1<=t<=k1
End_sync
kloop>=m*1
m:positive integer
[C1 => C2, C1 => C3 : Atomic Operation]

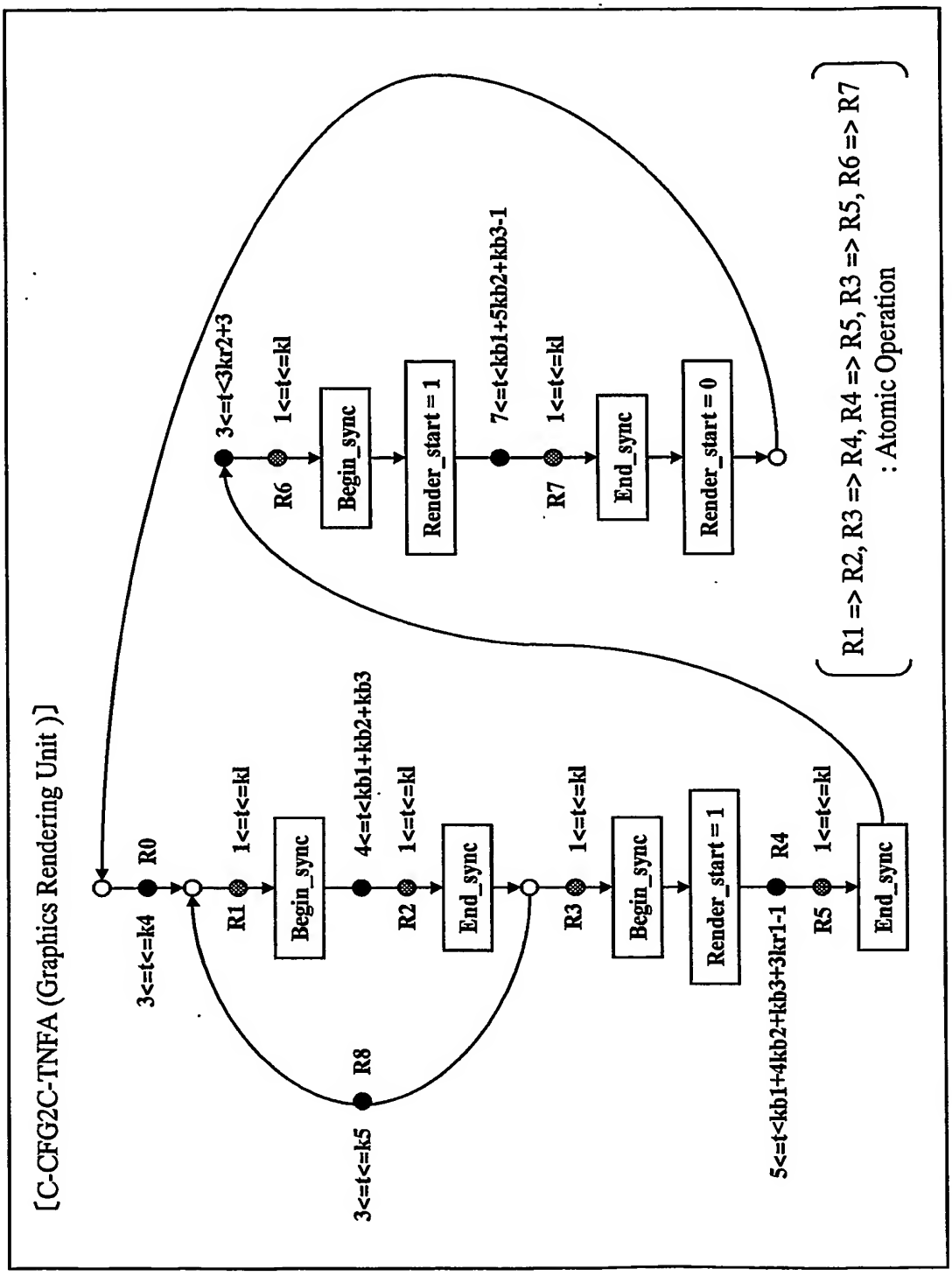
第６０図


[C-CFG2C-TNFA (Command Interface)]
C0
C1
C2
C3
C4
4<=t<=k1+k2+kloop1+4
5<=t<=k1+k2+k3+kloop1+5
7<=t<=kb1+3kb2+kb3+kl
sync
8<=t<=kb1+kb2+kb3+kl+3
sync
1<=t<=kl
1<=t<=kl
冗長ステート
kloop>m*1
m:positive integer

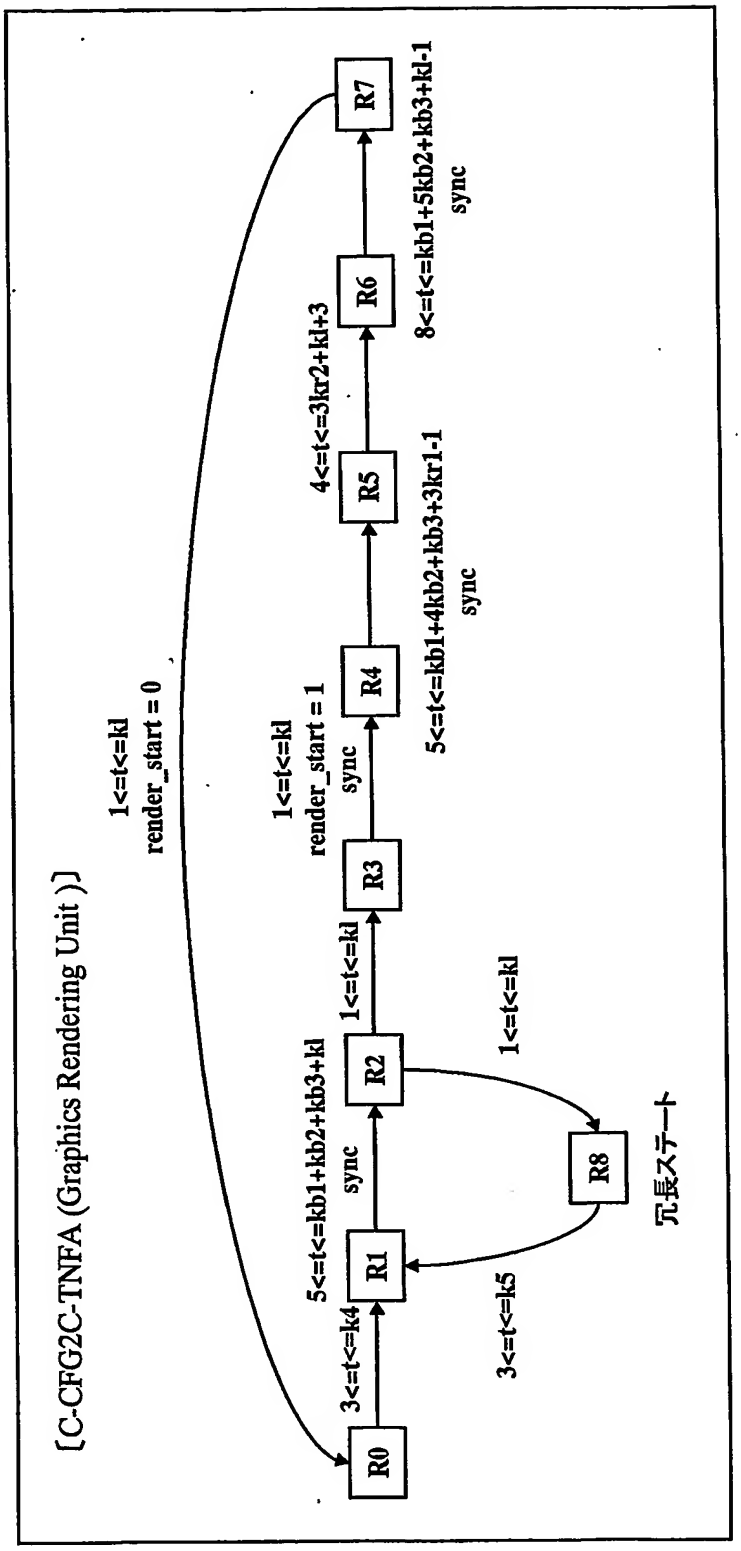
第６１図


[C-CFG2C-TNFA (Command Interface)]
C0
C1
C2
C3
4<=t<=k1+k2+kloop1+4
6<=t<=k1+k2+k3+kloop1+k1+5
7<=t<=kb1+3kb2+kb3+kl
sync
8<=t<=kb1+kb2+kb3+kl+3
sync
1<=t<=kl
kloop>m*1
m:positive integer

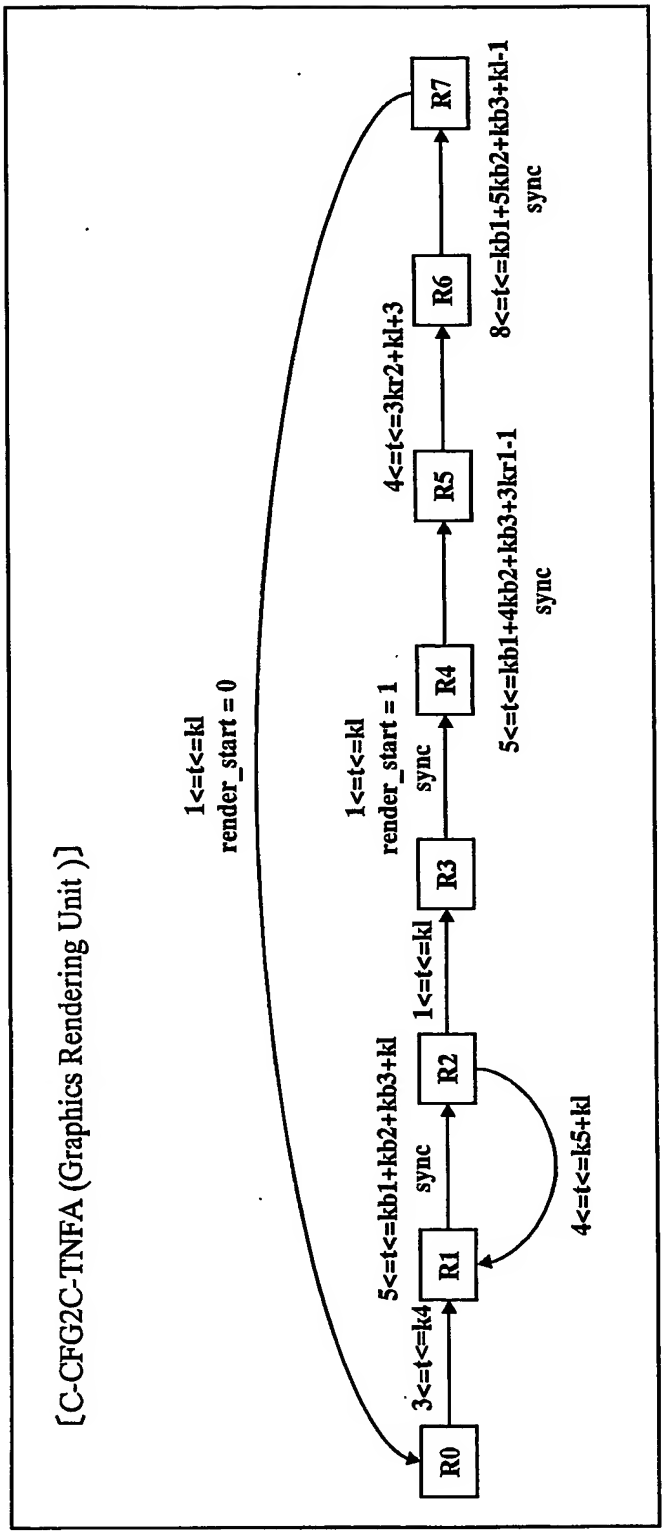
第62図


[C-CFG2C-TNFA (Graphics Rendering Unit)]
3<=t<=k4
R0
1<=t<=kl
R1
Begin_sync
4<=t<kb1+kb2+kb3
R2
1<=t<=kl
End_sync
3<=t<=k5
R8
1<=t<=kl
R3
Begin_sync
Render_start = 1
R4
5<=t<kb1+4kb2+kb3+3kr1-1
1<=t<=kl
R5
End_sync
3<=t<3kr2+3
1<=t<=kl
R6
Begin_sync
Render_start = 1
7<=t<kb1+5kb2+kb3-1
1<=t<=kl
R7
End_sync
Render_start = 0
R1 => R2, R3 => R4, R4 => R5, R3 => R5, R6 => R7
: Atomic Operation

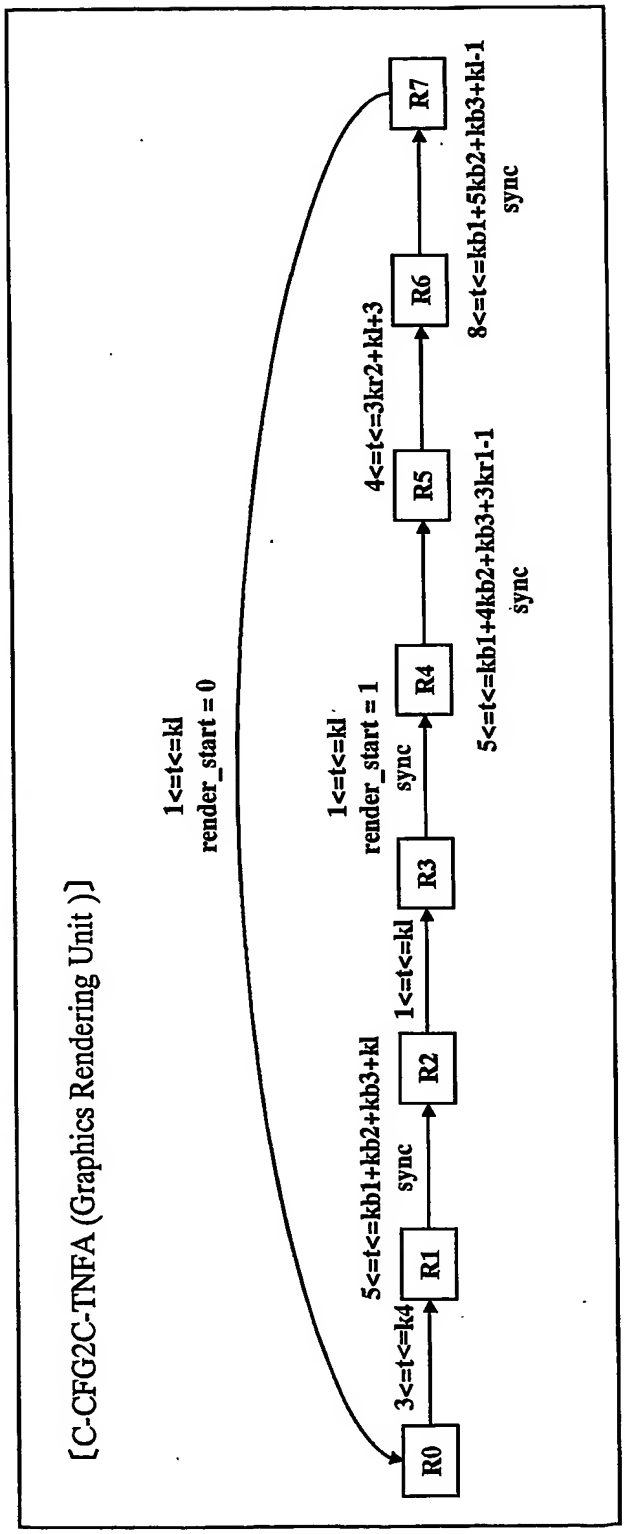
第６３図


〔C-CFG2C-TNFA (Graphics Rendering Unit)〕
R0
R1
R2
R3
R4
R5
R6
R7
R8
3<=t<=k4
5<=t<=kb1+kb2+kb3+kl
sync
1<=t<=kl
1<=t<=kl
render_start = 0
1<=t<=kl
render_start = 1
sync
5<=t<=kb1+4kb2+kb3+3kr1-1
sync
4<=t<=3kr2+kl+3
8<=t<=kb1+5kb2+kb3+kl-1
sync
1<=t<=kl
3<=t<=k5
冗長ステート

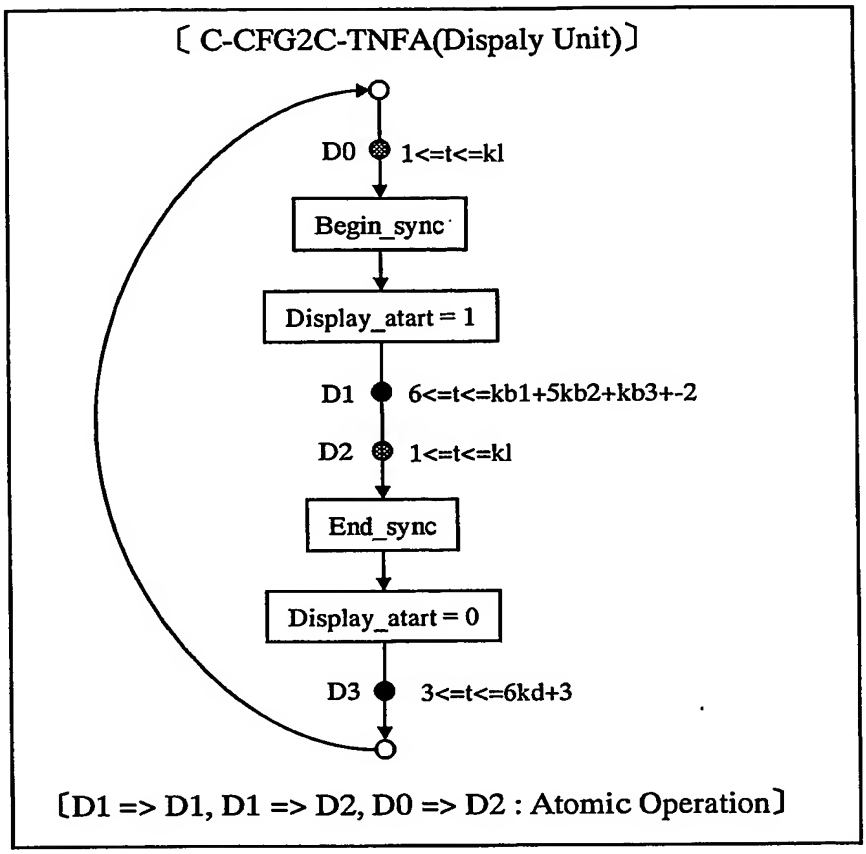
第64図


[C-CFG2C-TNFA (Graphics Rendering Unit)]
R0
R1
R2
R3
R4
R5
R6
R7
3<=t<=k4
5<=t<=kb1+kb2+kb3+kl
sync
1<=t<=kl
4<=t<=k5+kl
1<=t<=kl
render_start = 1
sync
5<=t<=kb1+4kb2+kb3+3kr1-1
sync
4<=t<=3kr2+kl+3
8<=t<=kb1+5kb2+kb3+kl-1
sync
1<=t<=kl
render_start = 0

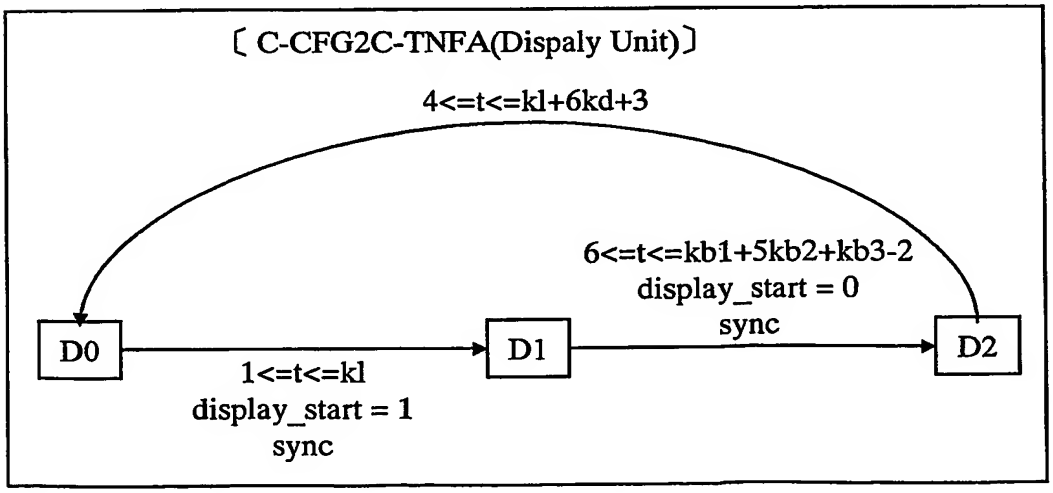
第65図


[C-CFG2C-TNFA (Graphics Rendering Unit)]
R0
R1
R2
R3
R4
R5
R6
R7
3<=t<=k4
5<=t<=kb1+kb2+kb3+kl
sync
1<=t<=kl
render_start = 0
1<=t<=kl
render_start = 1
sync
5<=t<=kb1+4kb2+kb3+3kr1-1
sync
4<=t<=3kr2+kl+3
8<=t<=kb1+5kb2+kb3+kl-1
sync

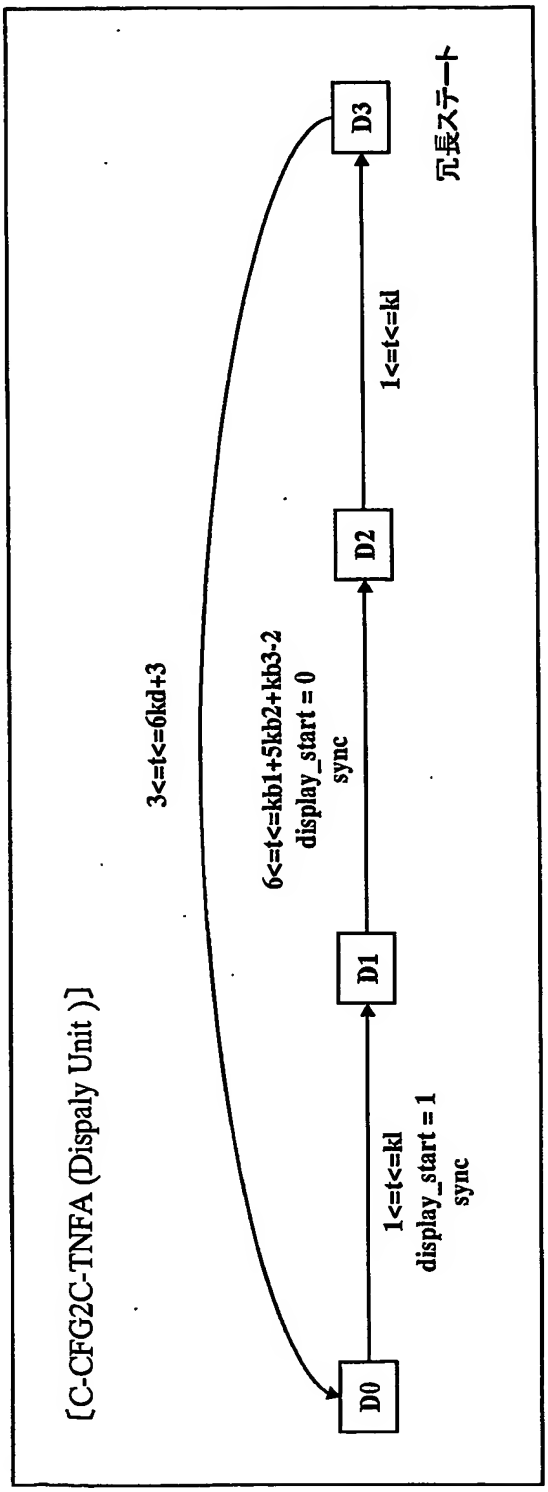
第６６図


〔C-CFG2C-TNFA(Dispaly Unit)〕
D0
1<=t<=kl
Begin_sync
Display_atart = 1
D1
6<=t<=kb1+5kb2+kb3+-2
D2
1<=t<=kl
End_sync
Display_atart = 0
D3
3<=t<=6kd+3
〔D1 => D1, D1 => D2, D0 => D2 : Atomic Operation〕

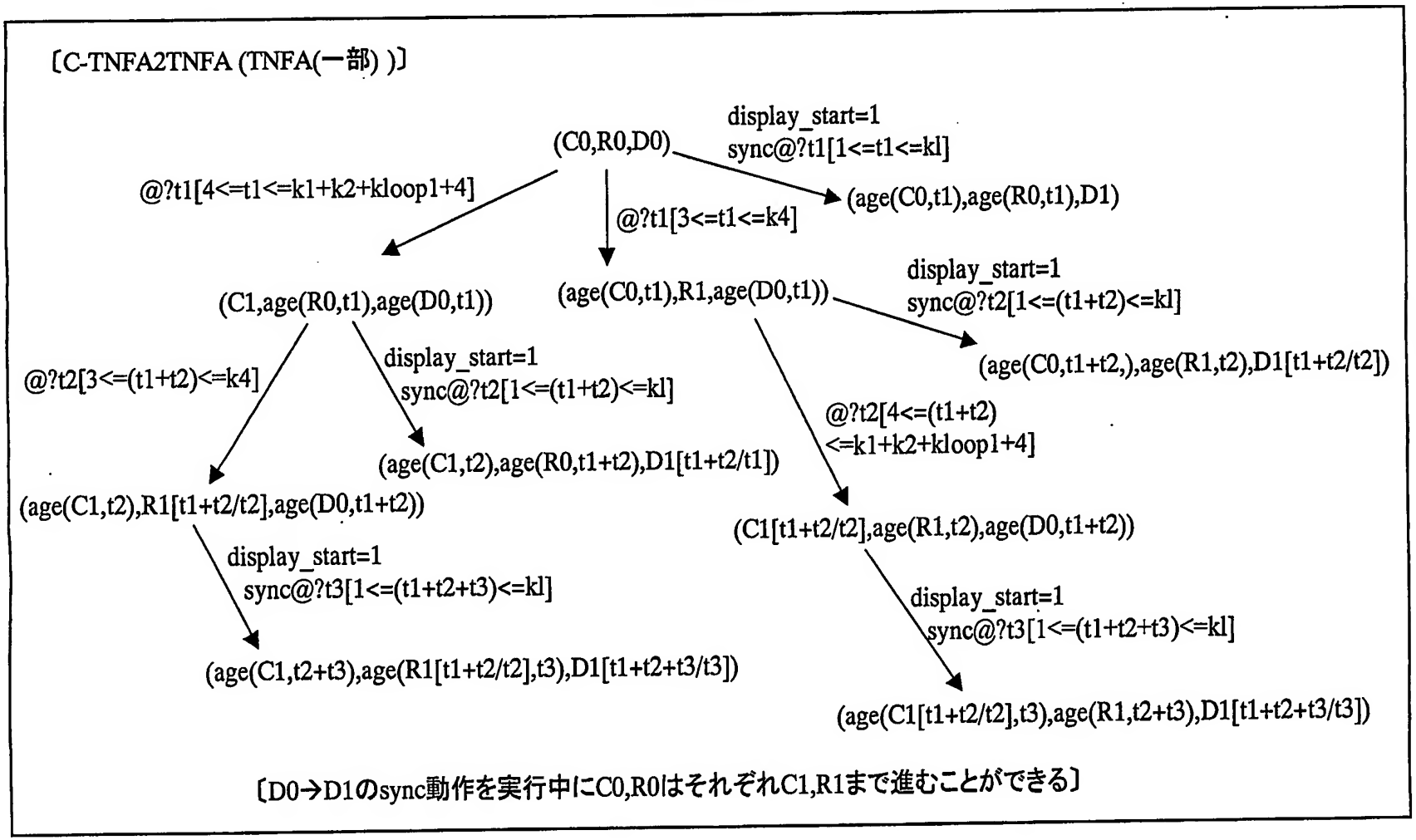
第６８図


〔C-CFG2C-TNFA(Dispaly Unit)〕
4<=t<=kl+6kd+3
6<=t<=kb1+5kb2+kb3-2
display_start = 0
sync
D0
D1
D2
1<=t<=kl
display_start = 1
sync

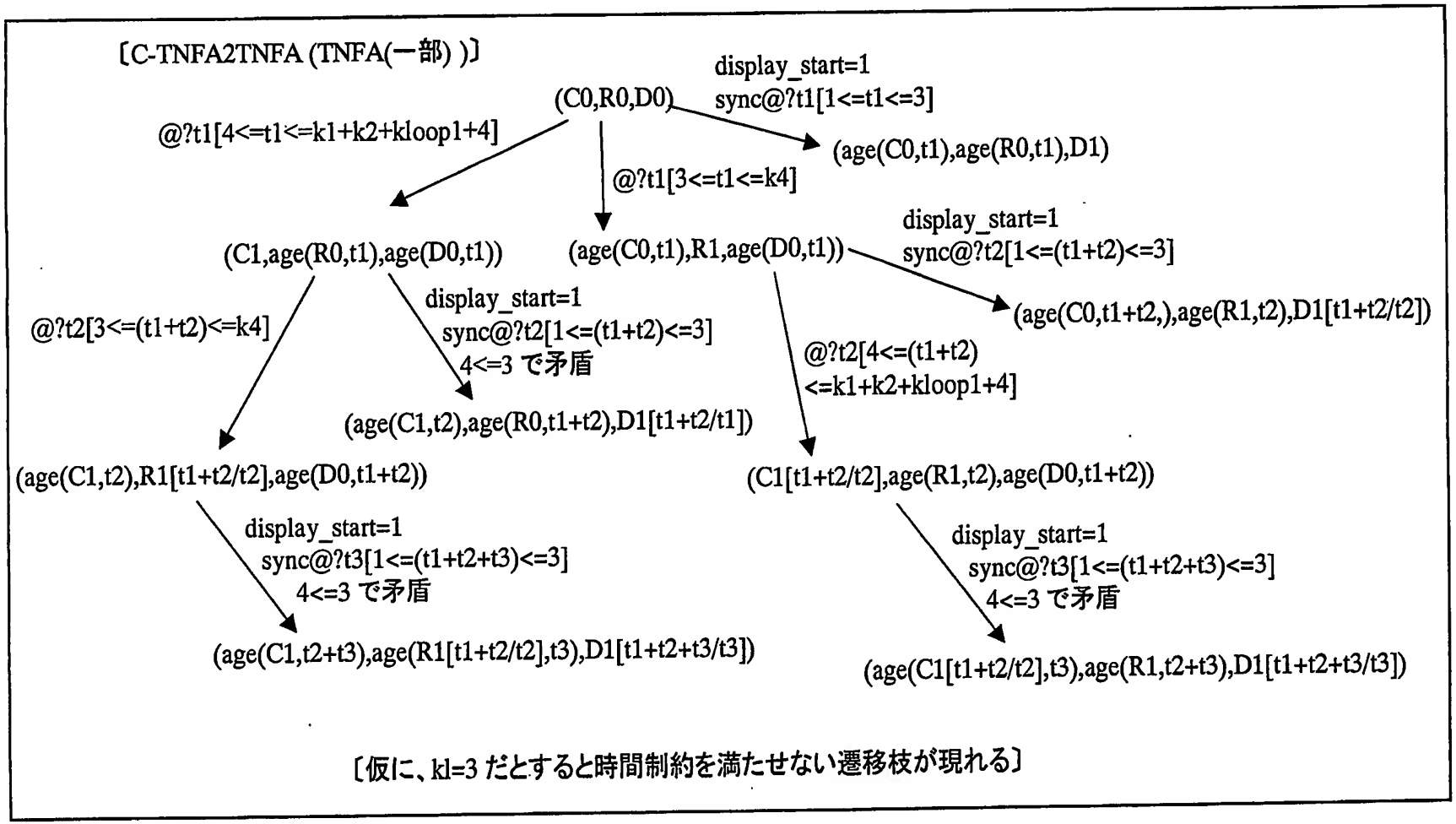
第６７図


[C-CFG2C-TNFA (Dispaly Unit)]
D0
D1
D2
D3
1<=t<=kl
display_start = 1
sync
6<=t<=kb1+5kb2+kb3-2
display_start = 0
sync
1<=t<=kl
3<=t<=6kd+3
冗長ステート

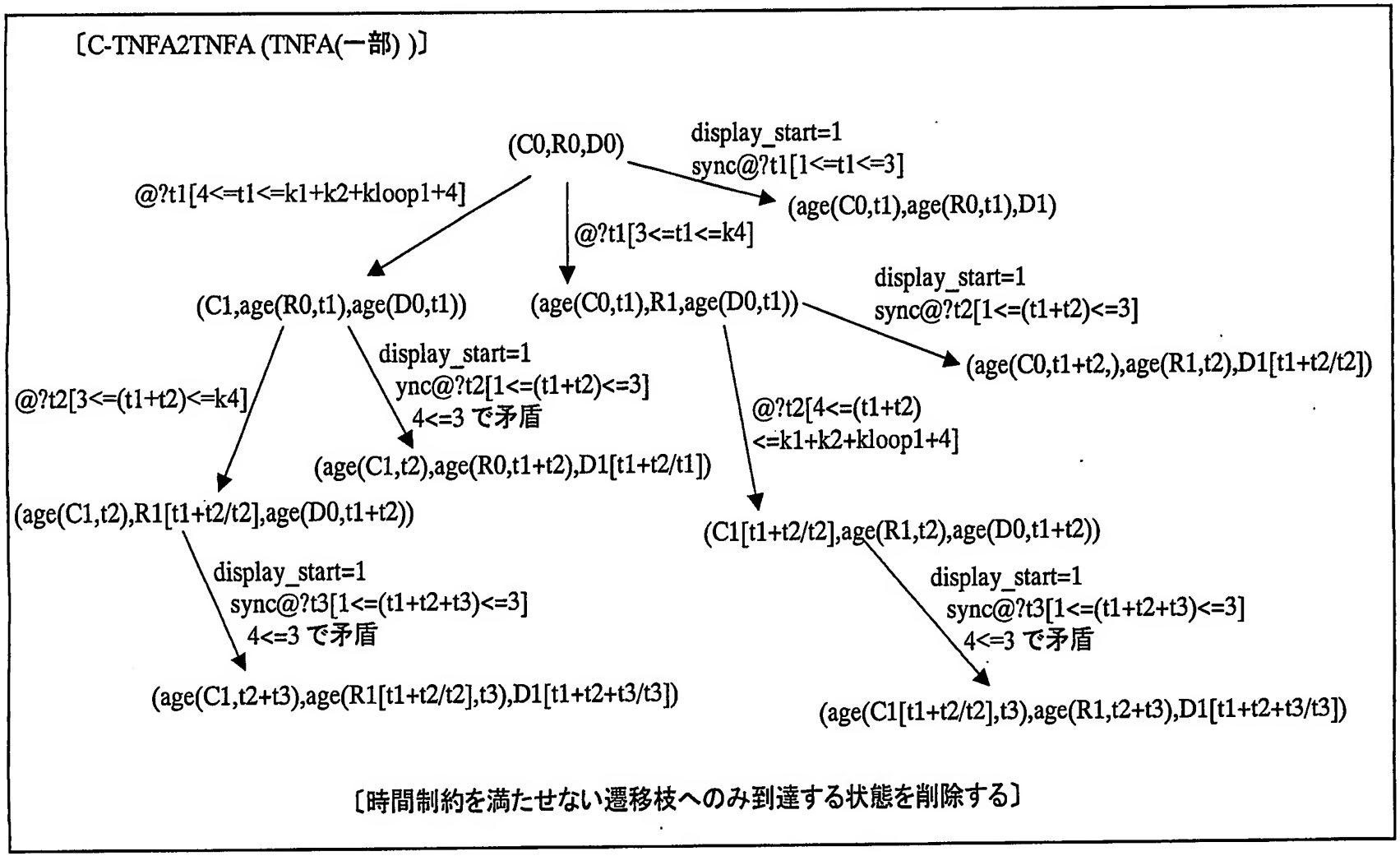
第６９図


〔C-TNFA2TNFA (TNFA(一部))〕
(C0,R0,D0)
display_start=1
sync@?t1[1<=t1<=kl]
(age(C0,t1),age(R0,t1),D1)
@?t1[4<=t1<=k1+k2+kloop1+4]
@?t1[3<=t1<=k4]
(C1,age(R0,t1),age(D0,t1))
(age(C0,t1),R1,age(D0,t1))
display_start=1
sync@?t2[1<=(t1+t2)<=kl]
(age(C0,t1+t2,),age(R1,t2),D1[t1+t2/t2])
@?t2[3<=(t1+t2)<=k4]
display_start=1
sync@?t2[1<=(t1+t2)<=kl]
(age(C1,t2),age(R0,t1+t2),D1[t1+t2/t1])
@?t2[4<=(t1+t2)
<=k1+k2+kloop1+4]
(age(C1,t2),R1[t1+t2/t2],age(D0,t1+t2))
(C1[t1+t2/t2],age(R1,t2),age(D0,t1+t2))
display_start=1
sync@?t3[1<=(t1+t2+t3)<=kl]
display_start=1
sync@?t3[1<=(t1+t2+t3)<=kl]
(age(C1,t2+t3),age(R1[t1+t2/t2],t3),D1[t1+t2+t3/t3])
(age(C1[t1+t2/t2],t3),age(R1,t2+t3),D1[t1+t2+t3/t3])
〔D0→D1のsync動作を実行中にC0,R0はそれぞれC1,R1まで進むことができる〕

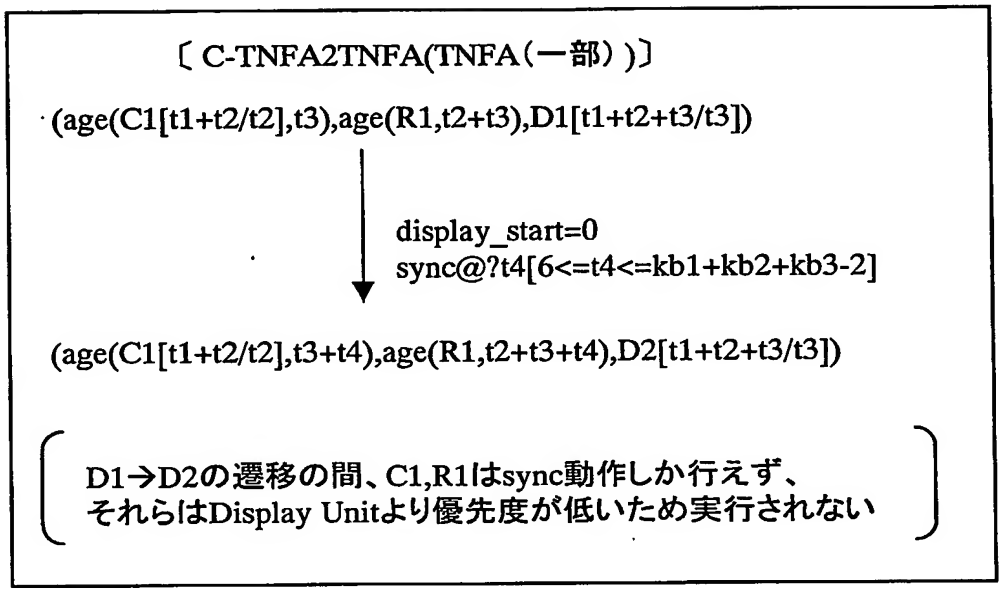
第７０図


〔C-TNFA2TNFA (TNFA(一部))〕
(C0,R0,D0)
display_start=1
sync@?t1[1<=t1<=3]
(age(C0,t1),age(R0,t1),D1)
@?t1[4<=t1<=k1+k2+kloop1+4]
@?t1[3<=t1<=k4]
(C1,age(R0,t1),age(D0,t1))
(age(C0,t1),R1,age(D0,t1))
display_start=1
sync@?t2[1<=(t1+t2)<=3]
(age(C0,t1+t2,),age(R1,t2),D1[t1+t2/t2])
@?t2[3<=(t1+t2)<=k4]
display_start=1
sync@?t2[1<=(t1+t2)<=3]
4<=3 で矛盾
(age(C1,t2),age(R0,t1+t2),D1[t1+t2/t1])
@?t2[4<=(t1+t2)
<=k1+k2+kloop1+4]
(age(C1,t2),R1[t1+t2/t2],age(D0,t1+t2))
(C1[t1+t2/t2],age(R1,t2),age(D0,t1+t2))
display_start=1
sync@?t3[1<=(t1+t2+t3)<=3]
4<=3 で矛盾
(age(C1,t2+t3),age(R1[t1+t2/t2],t3),D1[t1+t2+t3/t3])
display_start=1
sync@?t3[1<=(t1+t2+t3)<=3]
4<=3 で矛盾
(age(C1[t1+t2/t2],t3),age(R1,t2+t3),D1[t1+t2+t3/t3])
〔仮に、kl=3 だとすると時間制約を満たせない遷移枝が現れる〕

第７１図


〔C-TNFA2TNFA (TNFA(一部))〕
(C0,R0,D0)
display_start=1
sync@?t1[1<=t1<=3]
(age(C0,t1),age(R0,t1),D1)
@?t1[4<=t1<=k1+k2+kloop1+4]
@?t1[3<=t1<=k4]
(C1,age(R0,t1),age(D0,t1))
(age(C0,t1),R1,age(D0,t1))
display_start=1
sync@?t2[1<=(t1+t2)<=3]
(age(C0,t1+t2,),age(R1,t2),D1[t1+t2/t2])
display_start=1
ync@?t2[1<=(t1+t2)<=3]
4<=3 で矛盾
@?t2[3<=(t1+t2)<=k4]
(age(C1,t2),age(R0,t1+t2),D1[t1+t2/t1])
@?t2[4<=(t1+t2)
<=k1+k2+kloop1+4]
(age(C1,t2),R1[t1+t2/t2],age(D0,t1+t2))
(C1[t1+t2/t2],age(R1,t2),age(D0,t1+t2))
display_start=1
sync@?t3[1<=(t1+t2+t3)<=3]
4<=3 で矛盾
display_start=1
sync@?t3[1<=(t1+t2+t3)<=3]
4<=3 で矛盾
(age(C1,t2+t3),age(R1[t1+t2/t2],t3),D1[t1+t2+t3/t3])
(age(C1[t1+t2/t2],t3),age(R1,t2+t3),D1[t1+t2+t3/t3])
〔時間制約を満たせない遷移枝へのみ到達する状態を削除する〕

第７２図



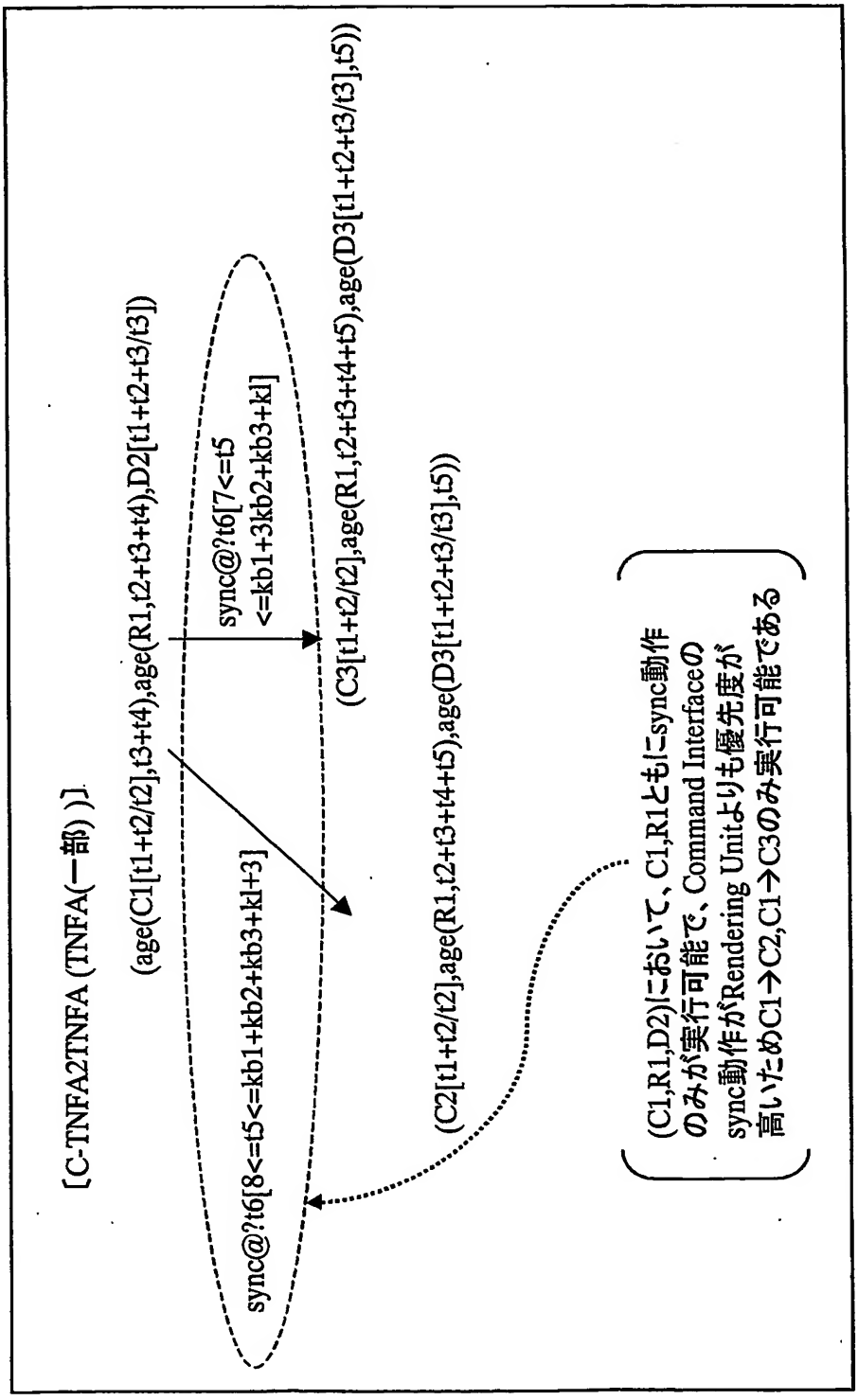
第８０図

〔パラメトリック解析結果(実行結果例)〕

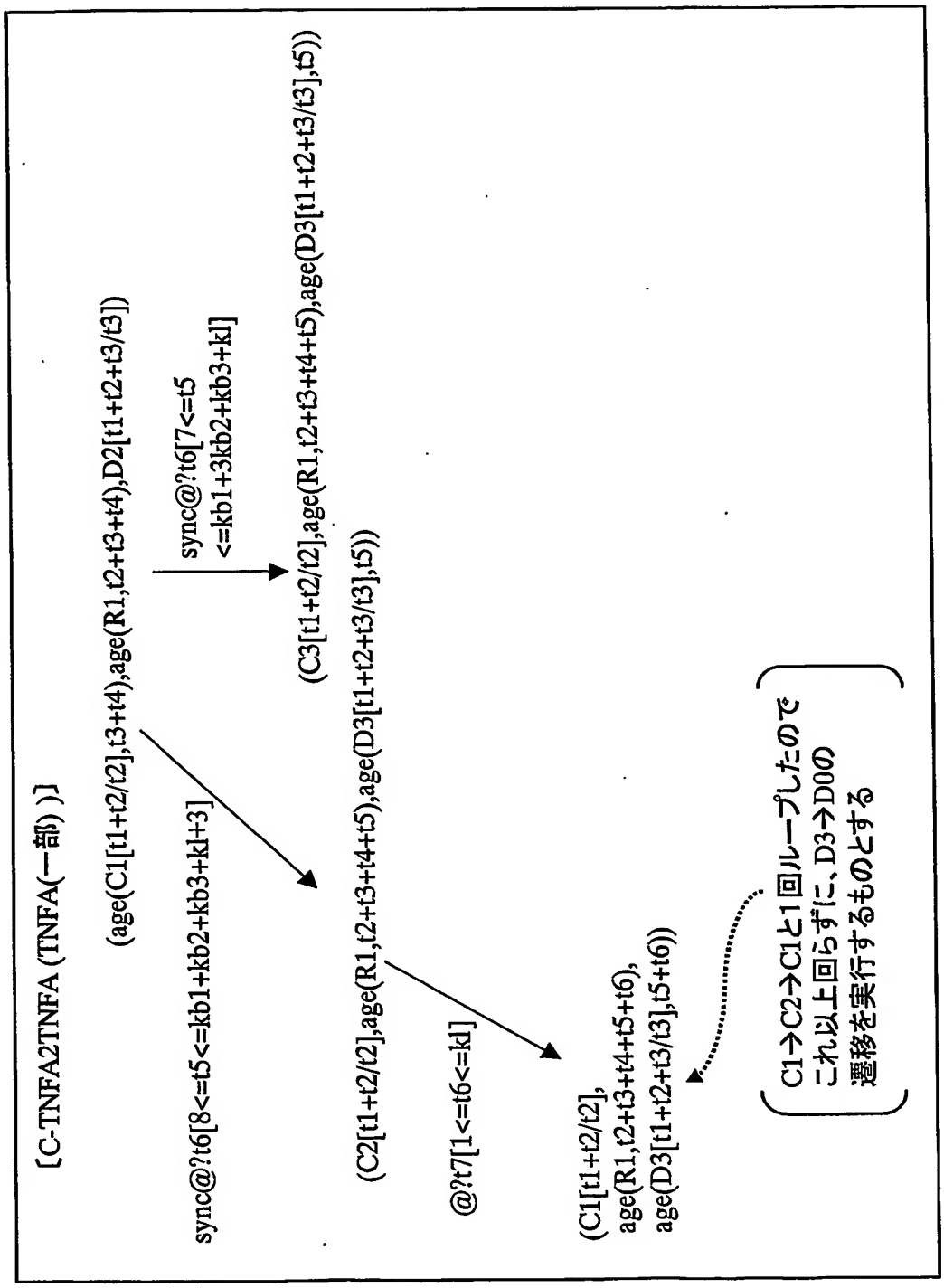
Value of objective function: 12

k1	0
k2	0
k3	0
k4	4
k5	0
kd	0
kl	4
kloop1	0
kb1	3
kb2	1
kb3	0
kr1	0
kr2	0

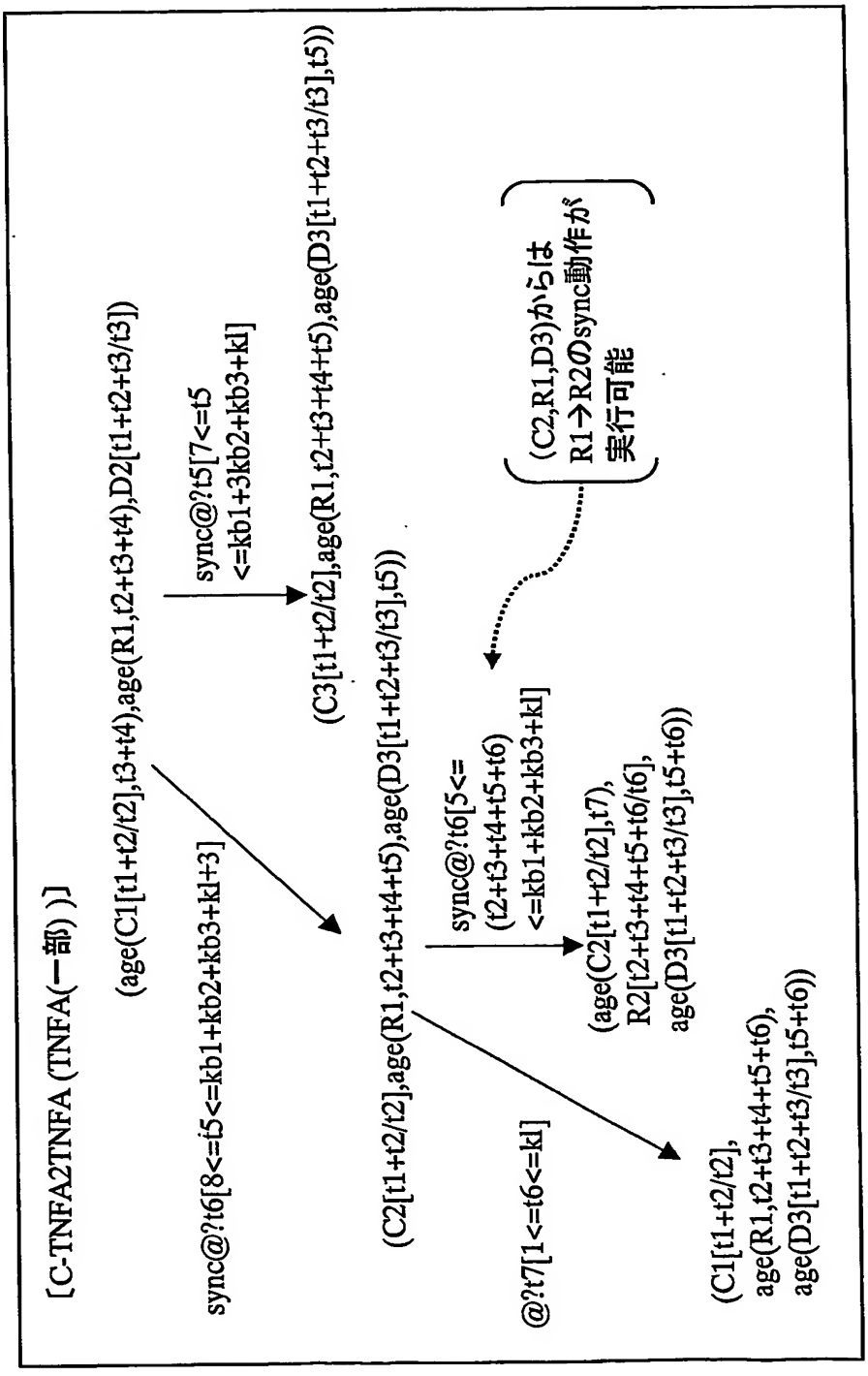
第７３図


[C-TNFA2TNFA (TNFA(一部))].
(age(C1[t1+t2/t2],t3+t4),age(R1,t2+t3+t4),D2[t1+t2+t3/t3])
sync@?t6[7<=t5
<=kb1+3kb2+kb3+kl]
(C3[t1+t2/t2],age(R1,t2+t3+t4+t5),age(D3[t1+t2+t3/t3],t5))
sync@?t6[8<=t5<=kb1+kb2+kb3+kl+3]
(C2[t1+t2/t2],age(R1,t2+t3+t4+t5),age(D3[t1+t2+t3/t3],t5))
(C1,R1,D2)において、C1,R1ともにsync動作
のみが実行可能で、Command Interfaceの
sync動作がRendering Unitよりも優先度が
高いためC1→C2,C1→C3のみ実行可能である

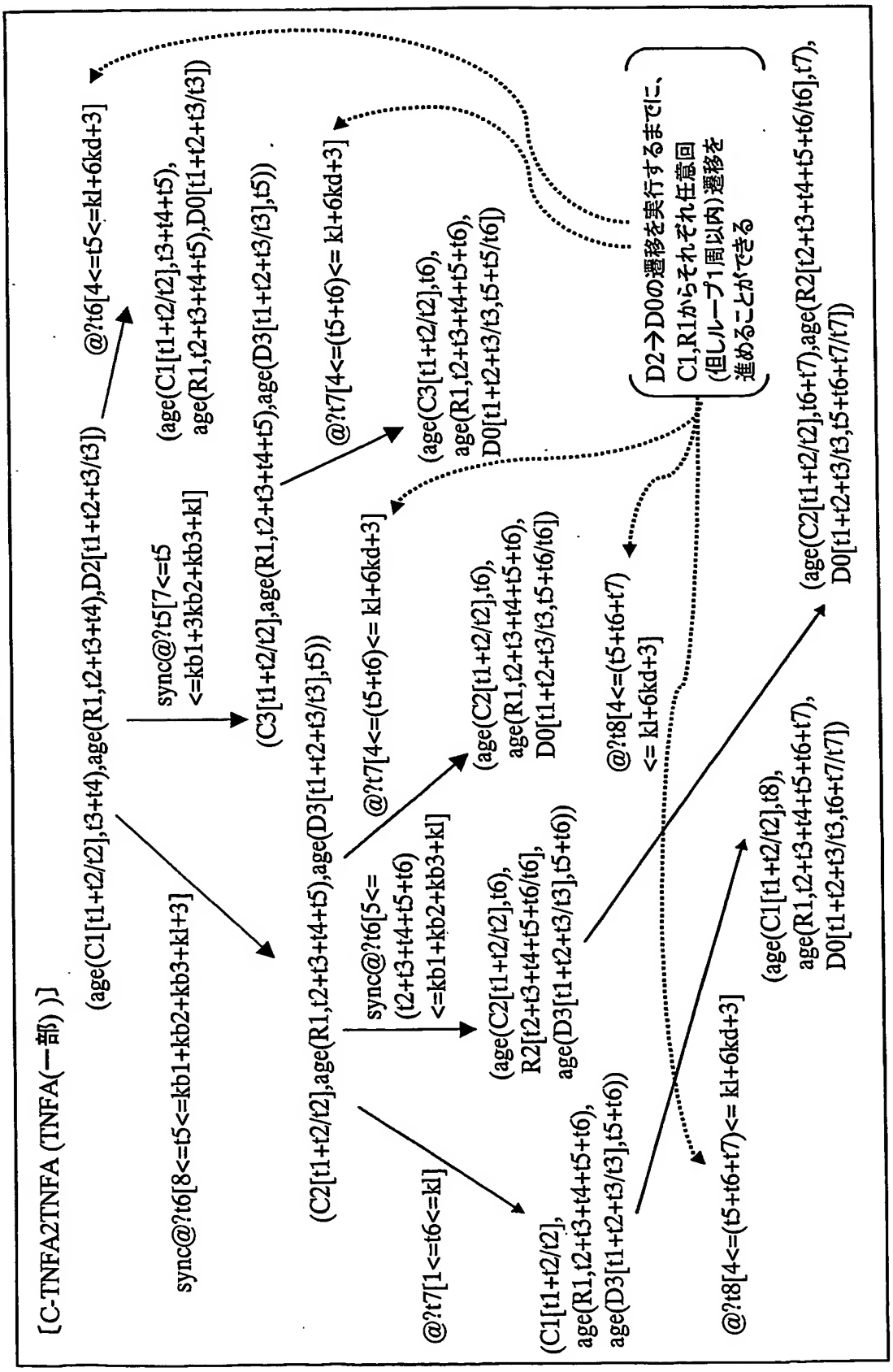
第７４図


〔C-TNFA2TNFA (TNFA(一部))〕
(age(C1[t1+t2/t2],t3+t4),age(R1,t2+t3+t4),D2[t1+t2+t3/t3])
sync@?t6[8<=t5<=kb1+kb2+kb3+kl+3]
sync@?t6[7<=t5
<=kb1+3kb2+kb3+kl]
(C3[t1+t2/t2],age(R1,t2+t3+t4+t5),age(D3[t1+t2+t3/t3],t5))
(C2[t1+t2/t2],age(R1,t2+t3+t4+t5),age(D3[t1+t2+t3/t3],t5))
@?t7[1<=t6<=kl]
(C1[t1+t2/t2],
age(R1,t2+t3+t4+t5+t6),
age(D3[t1+t2+t3/t3],t5+t6))
C1→C2→C1と1回ループしたので
これ以上回らずに、D3→D0の
遷移を実行するものとする

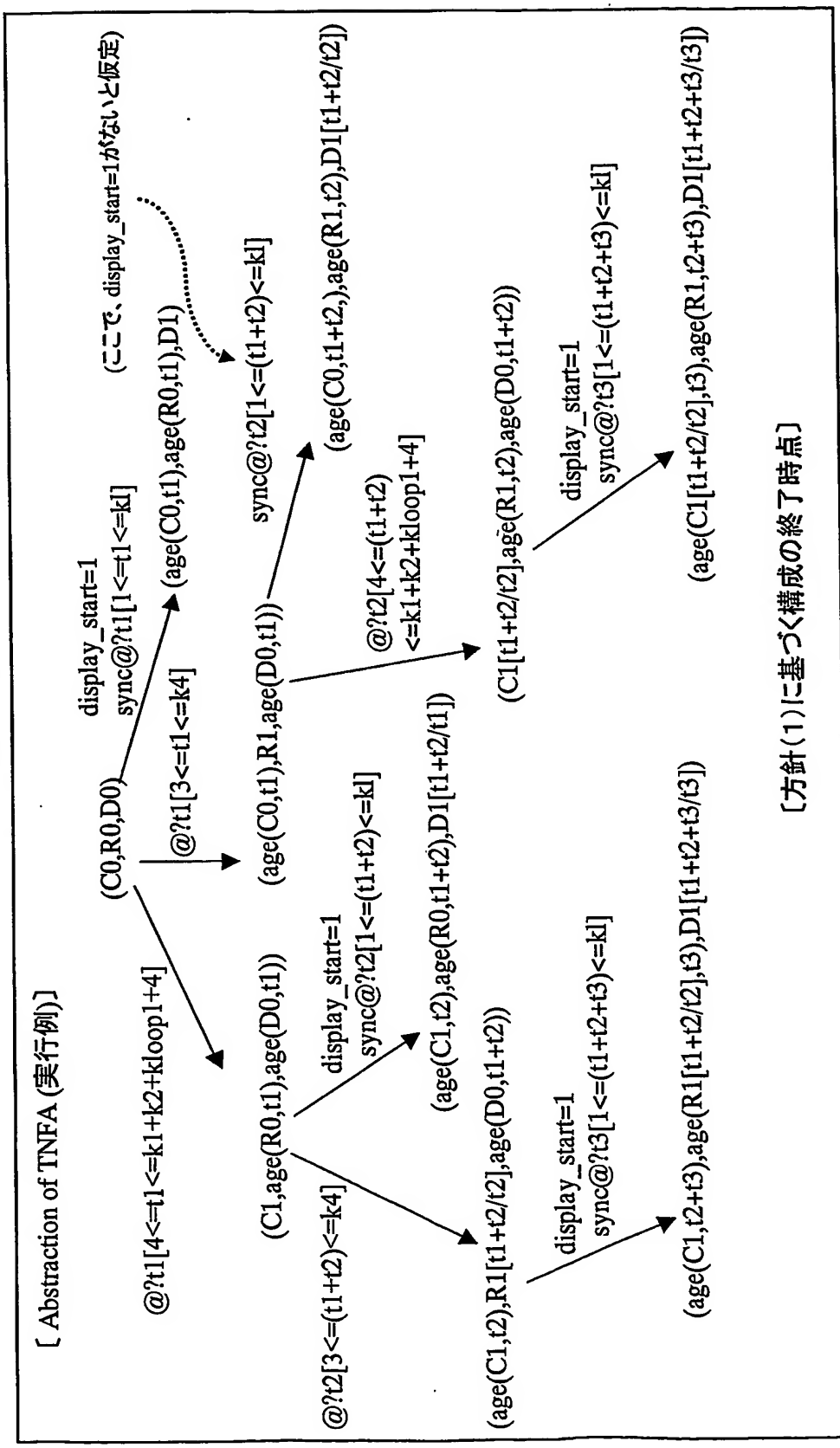
第７５図


[C-TNFA2TNFA (TNFA(一部))]
(age(C1[t1+t2/t2],t3+t4),age(R1,t2+t3+t4),D2[t1+t2+t3/t3])
sync@?t6[8<=t5<=kb1+kb2+kb3+kl+3]
sync@?t5[7<=t5
<=kb1+3kb2+kb3+kl]
(C3[t1+t2/t2],age(R1,t2+t3+t4+t5),age(D3[t1+t2+t3/t3],t5))
(C2[t1+t2/t2],age(R1,t2+t3+t4+t5),age(D3[t1+t2+t3/t3],t5))
sync@?t6[5<=
(t2+t3+t4+t5+t6)
<=kb1+kb2+kb3+kl]
@?t7[1<=t6<=kl]
(C2,R1,D3)からは
R1→R2のsync動作が
実行可能
(age(C2[t1+t2/t2],t7),
R2[t2+t3+t4+t5+t6/t6],
age(D3[t1+t2+t3/t3],t5+t6))
(C1[t1+t2/t2],
age(R1,t2+t3++t4++t5+t6),
age(D3[t1+t2+t3/t3],t5+t6))

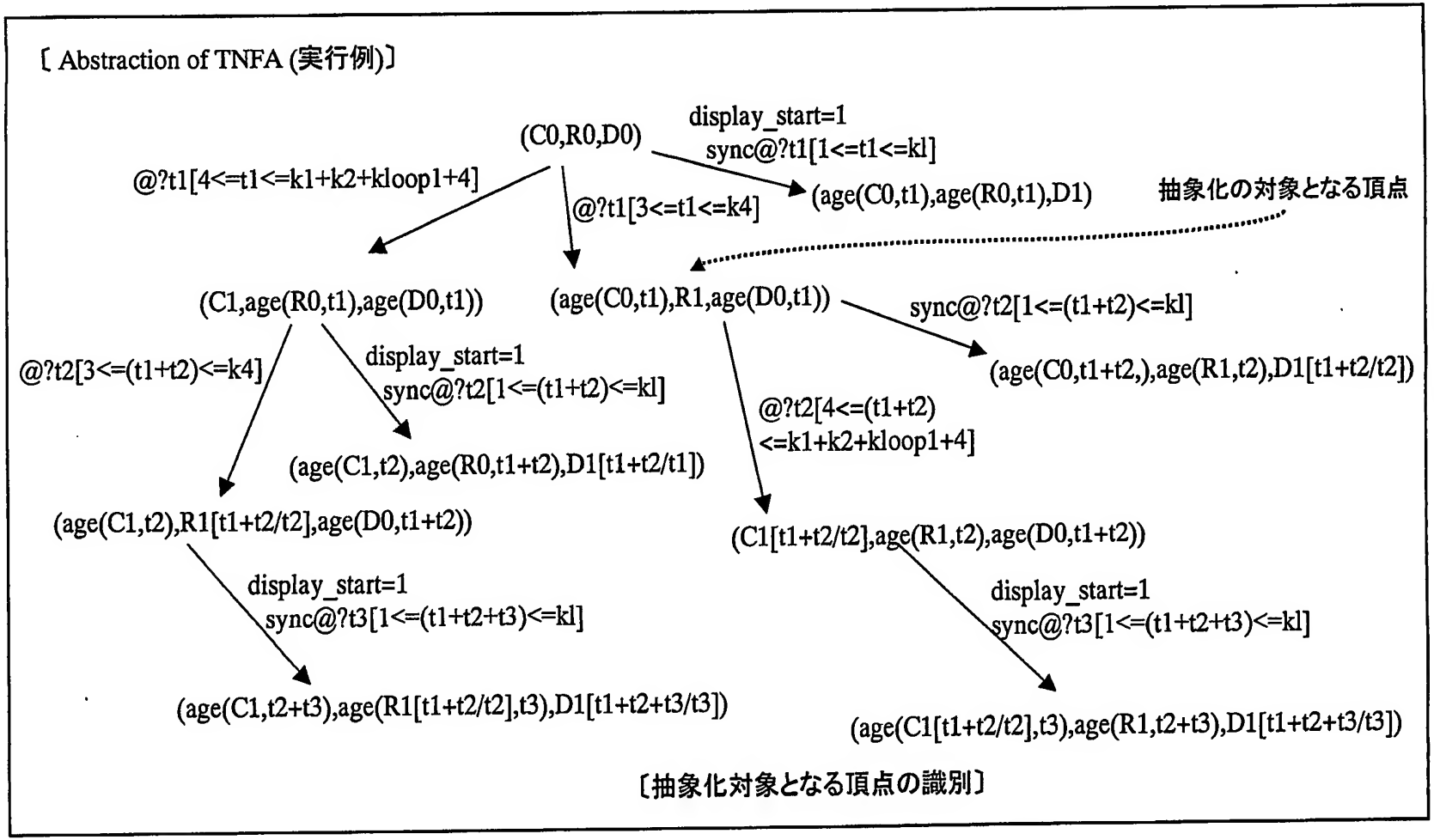
第７６図


[C-TNFA2TNFA (TNFA(一部))]
(age(C1[t1+t2/t2],t3+t4),age(R1,t2+t3+t4),D2[t1+t2+t3/t3])
@?t6[4<=t5<=kl+6kd+3]
sync@?t6[8<=t5<=kb1+kb2+kb3+kl+3]
sync@?t5[7<=t5
<=kb1+3kb2+kb3+kl]
(age(C1[t1+t2/t2],t3+t4+t5),
age(R1,t2+t3+t4+t5),D0[t1+t2+t3/t3])
(C3[t1+t2/t2],age(R1,t2+t3+t4+t5),age(D3[t1+t2+t3/t3],t5))
@?t7[4<=(t5+t6)<= kl+6kd+3]
(C2[t1+t2/t2],age(R1,t2+t3+t4+t5),age(D3[t1+t2+t3/t3],t5))
@?t7[4<=(t5+t6)<= kl+6kd+3]
(age(C3[t1+t2/t2],t6),
age(R1,t2+t3+t4+t5+t6),
D0[t1+t2+t3/t3,t5+t5/t6])
D2→D0の遷移を実行するまでに、
C1,R1からそれぞれ任意回
(但しループ1周以内)遷移を
進めることができる
@?t7[1<=t6<=kl]
sync@?t6[5<=
(t2+t3+t4+t5+t6)
<=kb1+kb2+kb3+kl]
(age(C2[t1+t2/t2],t6),
age(R1,t2+t3+t4+t5+t6),
D0[t1+t2+t3/t3,t5+t6/t6])
(C1[t1+t2/t2],
age(R1,t2+t3+t4+t5+t6),
age(D3[t1+t2+t3/t3],t5+t6))
(age(C2[t1+t2/t2],t6),
R2[t2+t3+t4+t5+t6/t6],
age(D3[t1+t2+t3/t3],t5+t6))
@?t8[4<=(t5+t6+f7)
<= kl+6kd+3]
@?t8[4<=(t5+t6+f7)<= kl+6kd+3]
(age(C1[t1+t2/t2],t8),
age(R1,t2+t3+t4+t5+t6+f7),
D0[t1+t2+t3/t3,t6+f7/f7])
(age(C2[t1+t2/t2],t6+f7),age(R2[t2+t3+t4+t5+t6/t6],f7),
D0[t1+t2+t3/t3,t5+t6+f7/f7])

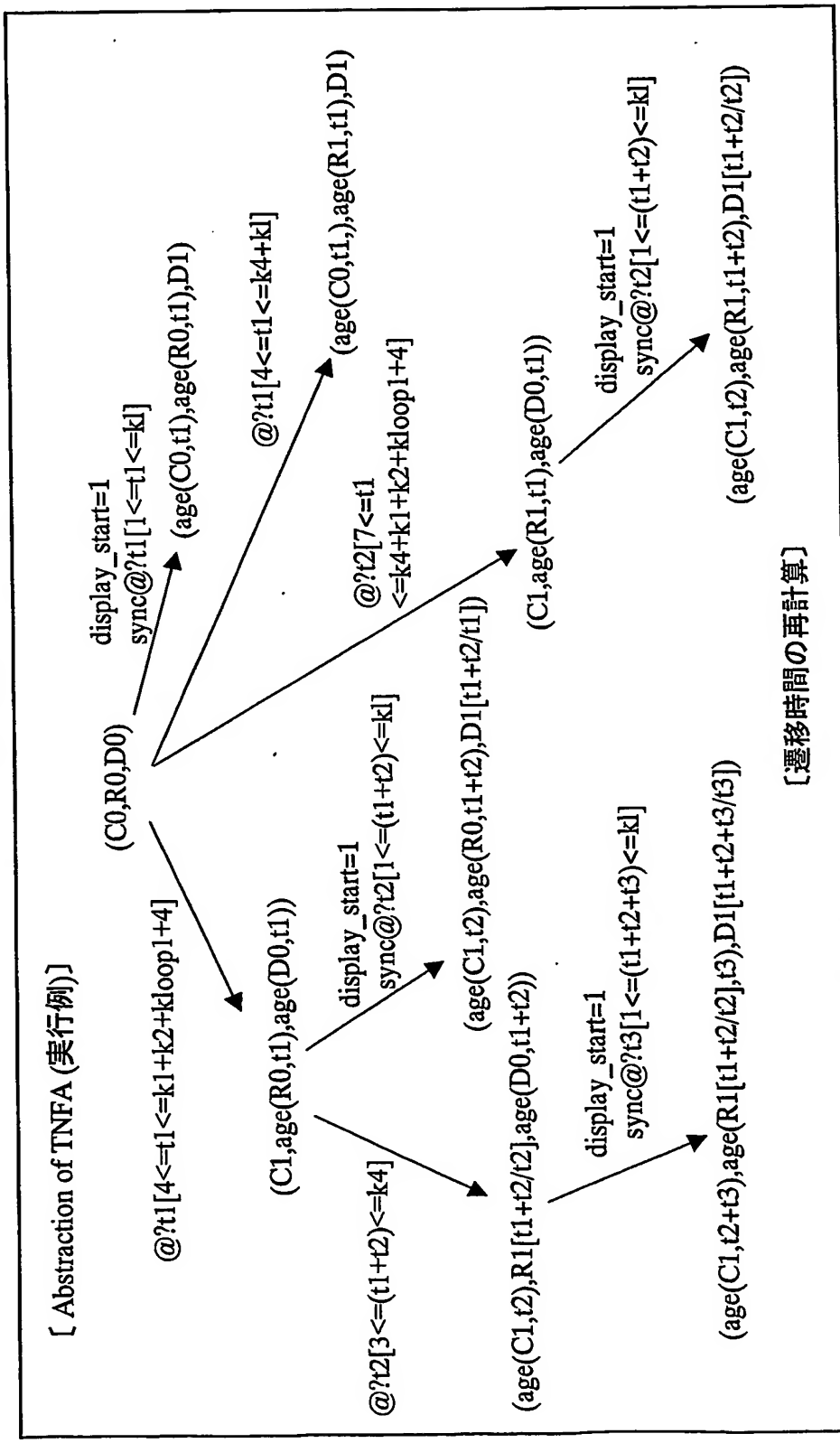
第７図


[Abstraction of TNFA (実行例)]
(C0,R0,D0)
@?t1[4<=t1<=k1+k2+kloop1+4]
display_start=1
sync@?t1[1<=t1<=kl]
(ここで、display_start=1がないと仮定)
@?t1[3<=t1<=k4]
(age(C0,t1),age(R0,t1),D1)
(C1,age(R0,t1),age(D0,t1))
@?t2[3<=(t1+t2)<=k4]
display_start=1
sync@?t2[1<=(t1+t2)<=kl]
(age(C0,t1),R1,age(D0,t1))
sync@?t2[1<=(t1+t2)<=kl]
(age(C0,t1+t2,),age(R1,t2),D1[t1+t2/t2])
@?t2[4<=(t1+t2)
<=k1+k2+kloop1+4]
(age(C1,t2),age(R0,t1+t2),D1[t1+t2/t1])
(age(C1,t2),R1[t1+t2/t2],age(D0,t1+t2))
(C1[t1+t2/t2],age(R1,t2),age(D0,t1+t2))
display_start=1
sync@?t3[1<=(t1+t2+t3)<=kl]
(age(C1,t2+t3),age(R1[t1+t2/t2],t3),D1[t1+t2+t3/t3])
display_start=1
sync@?t3[1<=(t1+t2+t3)<=kl]
(age(C1[t1+t2/t2],t3),age(R1,t2+t3),D1[t1+t2+t3/t3])
〔方針(1)に基づく構成の終了時点〕

第７８図


〔Abstraction of TNFA (実行例)〕
(C0,R0,D0)
display_start=1
sync@?t1[1<=t1<=kl]
@?t1[4<=t1<=k1+k2+kloop1+4]
@?t1[3<=t1<=k4]
(age(C0,t1),age(R0,t1),D1)
抽象化の対象となる頂点
(C1,age(R0,t1),age(D0,t1))
(age(C0,t1),R1,age(D0,t1))
sync@?t2[1<=(t1+t2)<=kl]
(age(C0,t1+t2,),age(R1,t2),D1[t1+t2/t2])
@?t2[3<=(t1+t2)<=k4]
display_start=1
sync@?t2[1<=(t1+t2)<=kl]
(age(C1,t2),age(R0,t1+t2),D1[t1+t2/t1])
@?t2[4<=(t1+t2)
<=k1+k2+kloop1+4]
(age(C1,t2),R1[t1+t2/t2],age(D0,t1+t2))
(C1[t1+t2/t2],age(R1,t2),age(D0,t1+t2))
display_start=1
sync@?t3[1<=(t1+t2+t3)<=kl]
display_start=1
sync@?t3[1<=(t1+t2+t3)<=kl]
(age(C1,t2+t3),age(R1[t1+t2/t2],t3),D1[t1+t2+t3/t3])
(age(C1[t1+t2/t2],t3),age(R1,t2+t3),D1[t1+t2+t3/t3])
〔抽象化対象となる頂点の識別〕

第７９図


〔Abstraction of TNFA (実行例)〕
(C0,R0,D0)
display_start=1
sync@?t1[1<=t1<=kl]
(age(C0,t1),age(R0,t1),D1)
@?t1[4<=t1<=k1+k2+kloop1+4]
@?t1[4<=t1<=k4+kl]
(age(C0,t1,),age(R1,t1),D1)
(C1,age(R0,t1),age(D0,t1))
display_start=1
sync@?t2[1<=(t1+t2)<=kl]
@?t2[7<=t1
<=k4+k1+k2+kloop1+4]
@?t2[3<=(t1+t2)<=k4]
(age(C1,t2),age(R0,t1+t2),D1[t1+t2/t1])
(C1,age(R1,t1),age(D0,t1))
(age(C1,t2),R1[t1+t2/t2],age(D0,t1+t2))
display_start=1
sync@?t2[1<=(t1+t2)<=kl]
display_start=1
sync@?t3[1<=(t1+t2+t3)<=kl]
(age(C1,t2),age(R1,t1+t2),D1[t1+t2/t2])
(age(C1,t2+t3),age(R1[t1+t2/t2],t3),D1[t1+t2+t3/t3])
〔遷移時間の再計算〕

第８１図

〔パラメトリック解析結果(実行結果例)〕

Value of objective function: 12

k1	0
k2	0
k3	0
k4	4
k5	0
kd	0
kl	4
kloop1	0
kb1	2
kb2	1
kb3	1
kr1	0
kr2	0

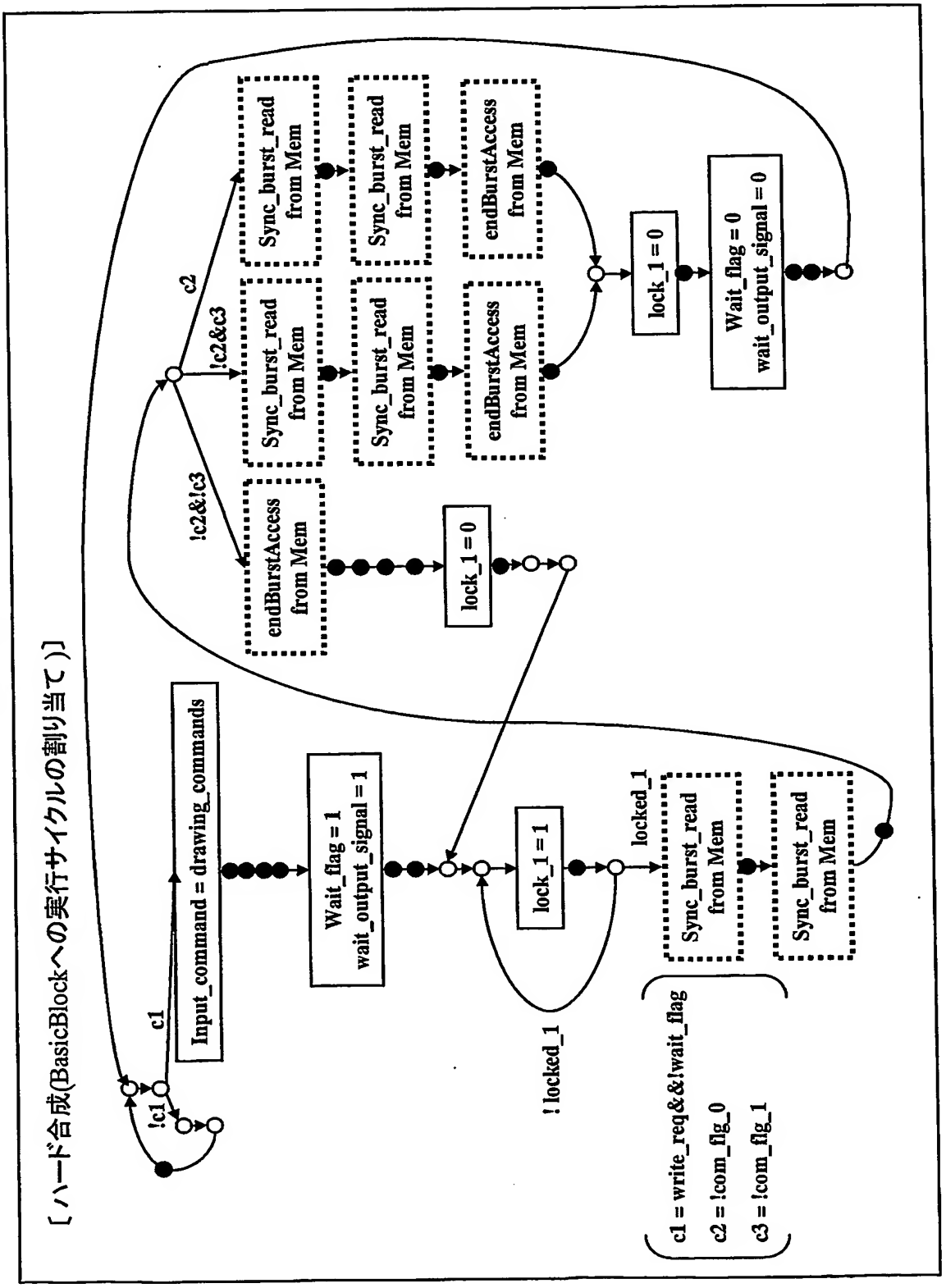
第８２図

〔パラメトリック解析結果(実行結果例)〕

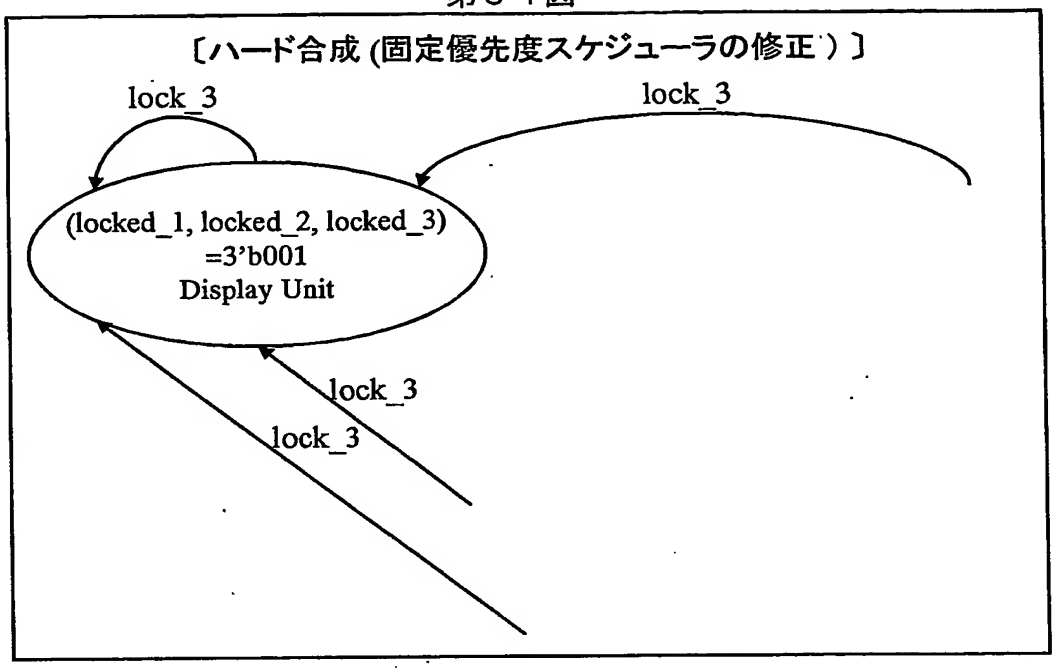
Value of objective function: 19

k1	1
k2	1
k3	1
k4	4
k5	1
kd	1
kl	4
kloop1	1
kb1	2
kb2	1
kb3	1
kr1	0
kr2	1

第８３図


〔ハード合成(BasicBlockへの実行サイクルの割り当て)〕
!c1
c1
Input_command = drawing_commands
Wait_flag = 1
wait_output_signal = 1
lock_1 = 1
! locked_1
locked_1
Sync_burst_read from Mem
Sync_burst_read from Mem
c1 = write_req&&!wait_flag
c2 = !com_flg_0
c3 = !com_flg_1
c2
!c2&c3
!c2&!c3
Sync_burst_read from Mem
Sync_burst_read from Mem
endBurstAccess from Mem
endBurstAccess from Mem
lock_1 = 0
lock_1 = 0
Wait_flag = 0
wait_output_signal = 0

第８４図

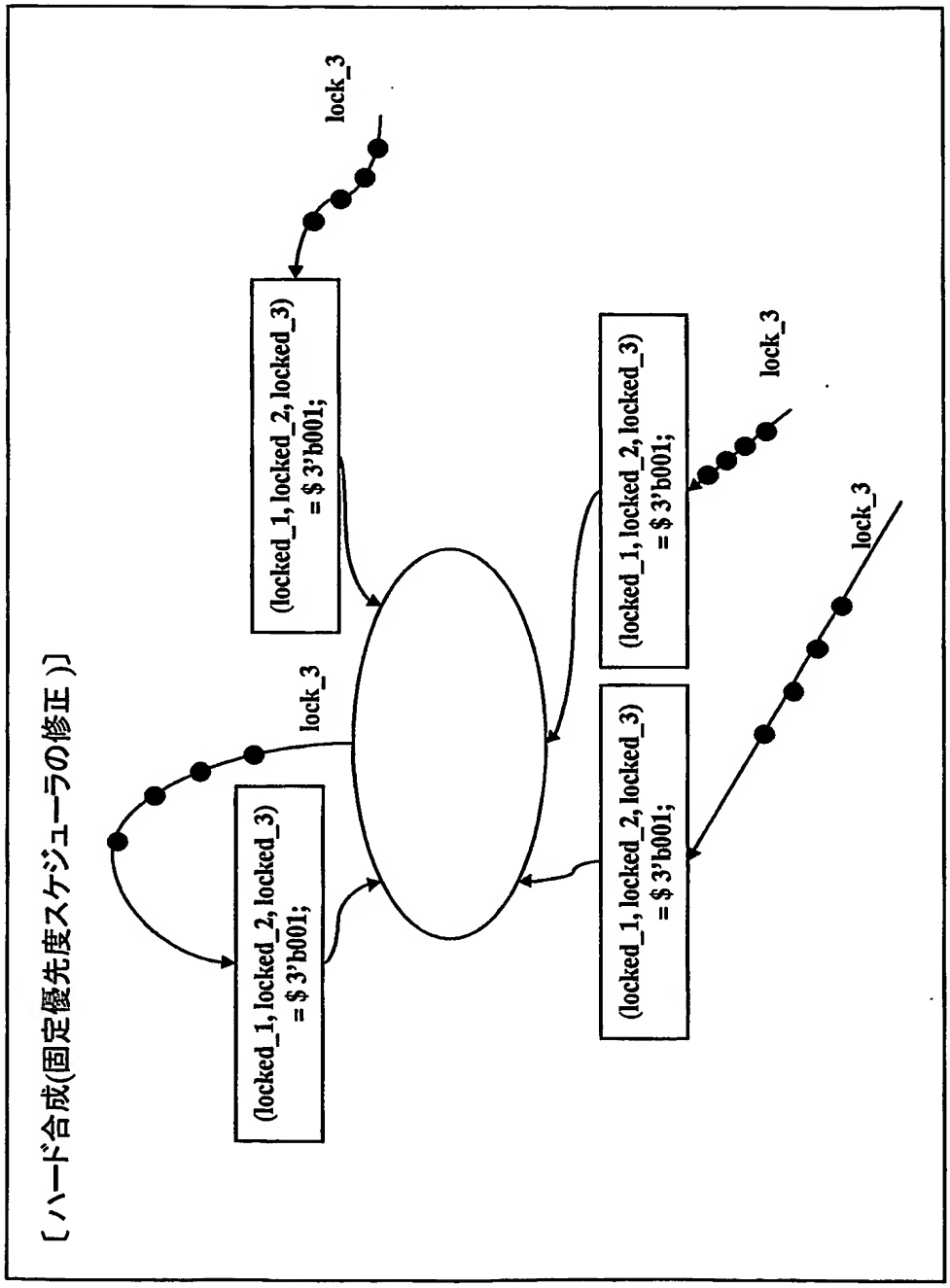


第９０図

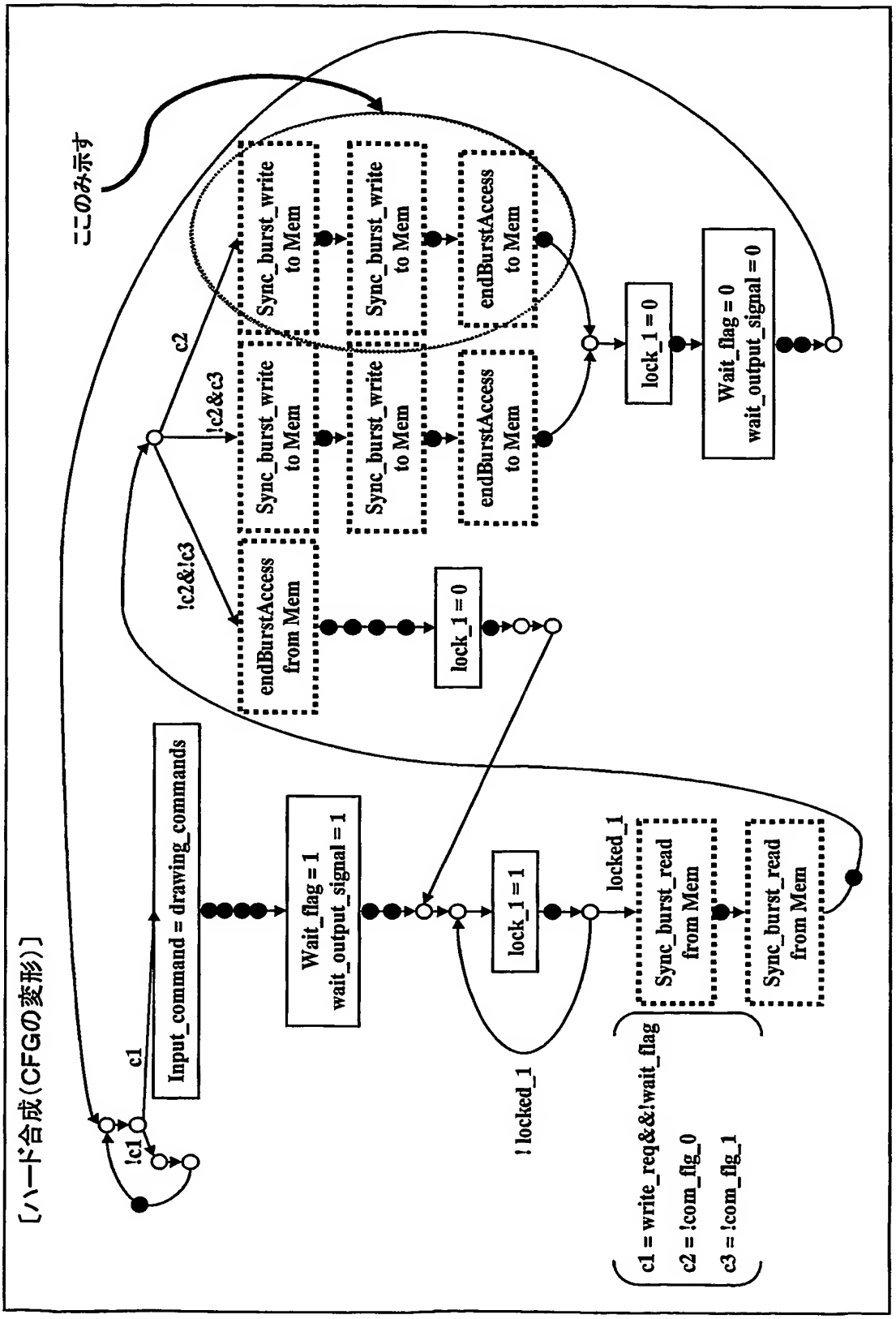
〔ハード合成 (共有レジスタ)〕

```
void register_in() {
  if (AD_BUS[3:0] == 4'b0000) {
    mem_con_reg.current_value[0] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0001) {
    mem_con_reg.current_value[1] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0010) {
    mem_con_reg.current_value[2] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0011) {
    mem_con_reg.current_value[3] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0100) {
    mem_con_reg.current_value[4] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0101) {
    mem_con_reg.current_value[5] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0110) {
    mem_con_reg.current_value[6] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b0111) {
    mem_con_reg.current_value[7] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b1000) {
    mem_con_reg.current_value[8] = $D_BUS;
  } else if (AD_BUS[3:0] == 4'b1001) {
    mem_con_reg.current_value[9] = $D_BUS;
  }
}
```

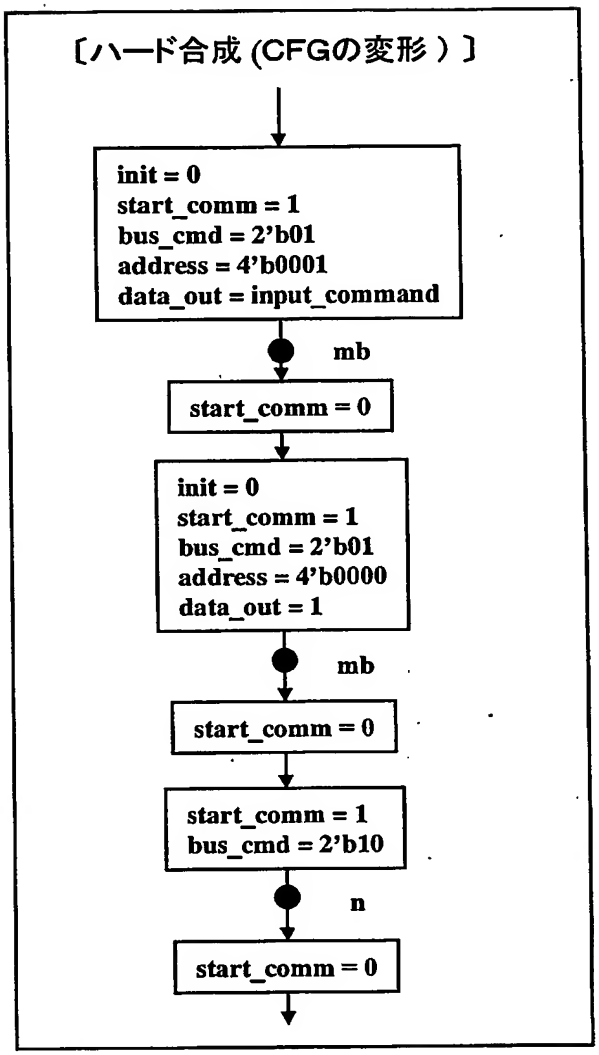
第８５図


〔ハード合成(固定優先度スケジューラの修正)〕
lock_3
(locked_1, locked_2, locked_3)
= $ 3'b001;
lock_3
(locked_1, locked_2, locked_3)
= $ 3'b001;
lock_3
(locked_1, locked_2, locked_3)
= $ 3'b001;
lock_3
(locked_1, locked_2, locked_3)
= $ 3'b001;

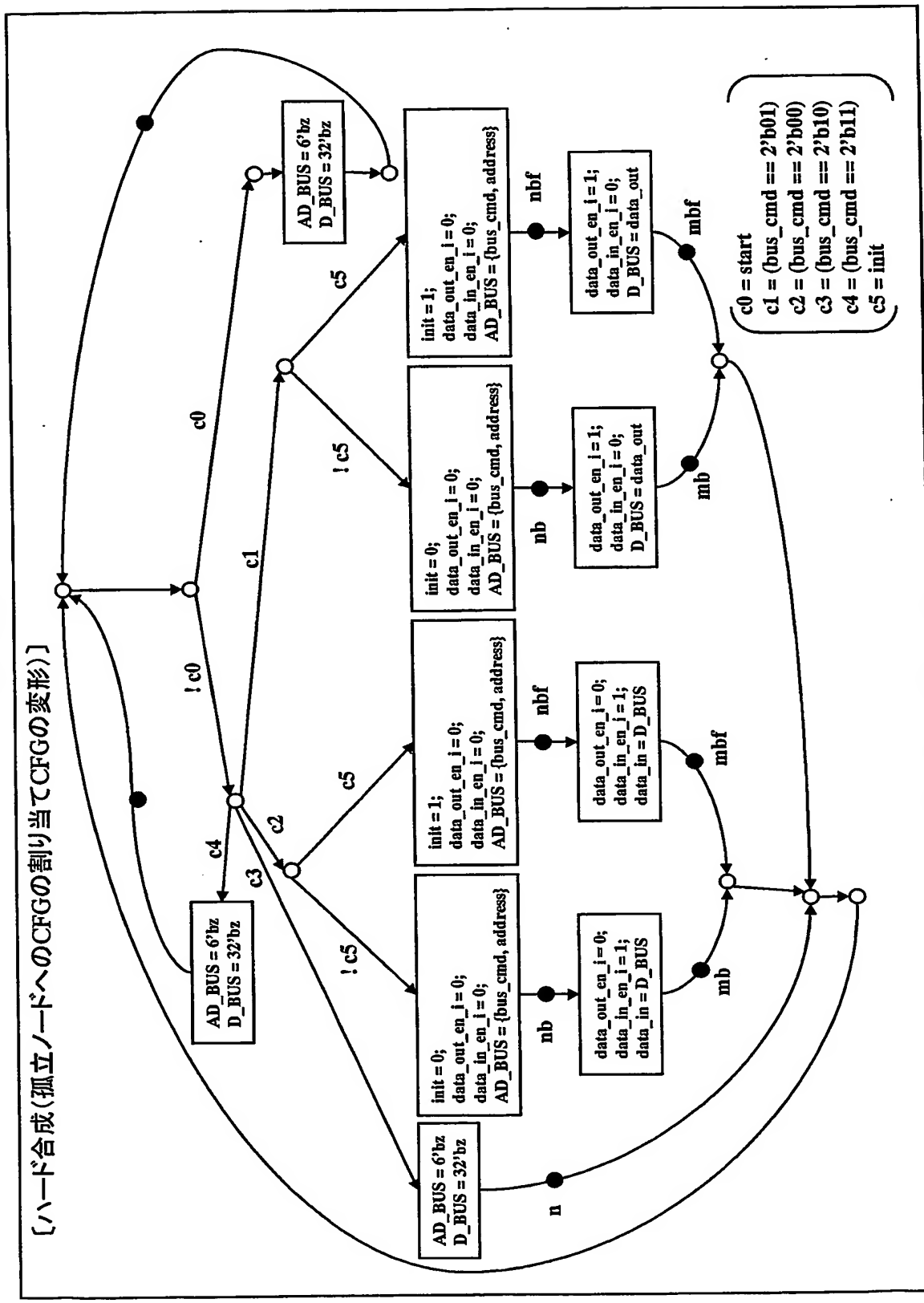
第８６図


〔ハード合成(CFGの変形)〕
ここのみ示す
!c1
c1
Input_command = drawing_commands
Wait_flag = 1
wait_output_signal = 1
lock_1 = 1
! locked_1
locked_1
Sync_burst_read
from Mem
Sync_burst_read
from Mem
c1 = write_req&&!wait_flag
c2 = !com_flg_0
c3 = !com_flg_1
!c2&!c3
endBurstAccess
from Mem
lock_1 = 0
c2
!c2&c3
Sync_burst_write
to Mem
Sync_burst_write
to Mem
endBurstAccess
to Mem
Sync_burst_write
to Mem
Sync_burst_write
to Mem
endBurstAccess
to Mem
lock_1 = 0
Wait_flag = 0
wait_output_signal = 0

第８７図


〔ハード合成 (CFGの変形) 〕
init = 0
start_comm = 1
bus_cmd = 2'b01
address = 4'b0001
data_out = input_command
mb
start_comm = 0
init = 0
start_comm = 1
bus_cmd = 2'b01
address = 4'b0000
data_out = 1
mb
start_comm = 0
start_comm = 1
bus_cmd = 2'b10
n
start_comm = 0

第８８図


〔ハード合成(孤立ノードへのCFGの割り当てCFGの変形)〕
c0 = start
c1 = (bus_cmd == 2'b01)
c2 = (bus_cmd == 2'b00)
c3 = (bus_cmd == 2'b10)
c4 = (bus_cmd == 2'b11)
c5 = init
AD_BUS = 6'bz
D_BUS = 32'bz
init = 1;
data_out_en_i = 0;
data_in_en_i = 0;
AD_BUS = {bus_cmd, address}
init = 0;
data_out_en_i = 0;
data_in_en_i = 0;
AD_BUS = {bus_cmd, address}
data_out_en_i = 0;
data_in_en_i = 1;
data_in = D_BUS
data_out_en_i = 1;
data_in_en_i = 0;
D_BUS = data_out
c0
! c0
c1
! c5
c5
c2
c3
c4
n
nb
nbf
mb
mbf

第８９図

〔ハード合成（共有レジスタ）〕

```
while(1) {
 D_BUS = 32'bz;
 if (locked_1 || locked_2 || locked_3) {
  if (AD_BUS[5:4] == 2'b00) {
   if (init) {
    L1 :
    if (data_out_en_1 || data_out_en_2 || data_out_en_3) {
     $
     register_in();
    } else {
     $ goto L1;
    }
   } else {
    L2 :
    if (data_out_en_1 || data_out_en_2 || data_out_en_3) {
     register_in();
    } else {
     $ goto L2;
    }
   }
  } else if (AD_BUS == 2'b01) {
   if (init) {
    L3 :
    if (data_in_en_1 || data_in_en_2 || data_in_en_3) {
     $
     register_out();
    } else {
     $ goto L3;
    }    } else {
    L4 :
    if (data_in_en_1 || data_in_en_2 || data_in_en_3) {
     register_out();
    } else {
     $ goto L4;
    }
   }
  }
  $
}
```


CT/JP2003/012840

第９１図

〔ハード合成 (共有レジスタ)〕

```
void register_out() {
  if (AD_BUS[3:0] == 4'b0000) {
    D_BUS = mem_con_reg.current_value[0];
  } else if (AD_BUS[3:0] == 4'b0001) {
    D_BUS = mem_con_reg.current_value[1];
  } else if (AD_BUS[3:0] == 4'b0010) {
    D_BUS = mem_con_reg.current_value[2];
  } else if (AD_BUS[3:0] == 4'b0011) {
    D_BUS = mem_con_reg.current_value[3];
  } else if (AD_BUS[3:0] == 4'b0100) {
    D_BUS = mem_con_reg.current_value[4];
  } else if (AD_BUS[3:0] == 4'b0101) {
    D_BUS = mem_con_reg.current_value[5];
  } else if (AD_BUS[3:0] == 4'b0110) {
    D_BUS = mem_con_reg.current_value[6];
  } else if (AD_BUS[3:0] == 4'b0111) {
    D_BUS = mem_con_reg.current_value[7];
  } else if (AD_BUS[3:0] == 4'b1000) {
    D_BUS = mem_con_reg.current_value[8];
  } else if (AD_BUS[3:0] == 4'b1001) {
    D_BUS = mem_con_reg.current_value[9];
  }
}
```

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP03/12840

A. CLASSIFICATION OF SUBJECT MATTER
Int.Cl[7] G06F17/50

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
Int.Cl[7] G06F17/50

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	Masayuki IENAGA et al., "Seigyo Shori Hardware no Koi Gosei no Tame no Kosoku na Menseki/Jikan Saitekika Algorithm", DA Symposium 2000, Information Processing Society of Japan, 17 July, 2000 (17.07.00), Vol.2000, No.8, pages 27 to 32	1-4,6
Y	NAKATA, A. et al., "Deriving Parameter Conditions for Periodic Timed Automata Satisfying Real-Time Temporal Logic Formulas", Proc. of IFIP TC6/WG6. 1 Int.Conf. on Formal Techniques for Networked and Distributed Systems (FORTE2001), Kluwer Academic Publishers, 2001.08, pages 151 to 166	1-4,6
Y	GB 2317245 A (SHARP KABUSHIKI KAISHA), 18 March, 1998 (18.03.98), Full text & JP 10-116302 A	1-4,6

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
06 November, 2003 (06.11.03)	25 November, 2003 (25.11.03)
Name and mailing address of the ISA/ Japanese Patent Office	Authorized officer
Facsimile No.	Telephone No.

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP03/12840

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	Kazutoshi WAKABAYASHI et al., "Densoyo LSI o Dosa Gosei de Kaihatsu, Kino Sekkei no Kikan ga 1/10 ni Tanshuku", Nikkei Electronics, Nikkei Business Publications, Inc., 12 February, 1996 (12.02.96), No.655, pages 147 to 169	1-4,6
A	Hitoshi OKEWATARI et al., "Multi Processor ni yoru Bunsan Shori o Ishiki Shita Senyo Processor Sekkei Shien System SYARDS no Kochiku", Information Processing Society of Japan Kenkyu Hokoku, Information Processing Society of Japan, 20 January, 1995 (20.01.95), Vol.95, No.6(DA-73), pages 105 to 112	5
P,X	Satoru KITAGUCHI et al., "Jitsujikan Seiyaku o Yusuru Tan'itsu Bus System no JAVA ni yoru Model-ka oyobi Parametric Model Checking o Mochiita Sekkei Shuho no Teian", Information Processing Society of Japan Kenkyu Hokoku, Information Processing Society of Japan, 28 November, 2002 (28.11.02), Vol.2002, No.113(SLDM-107), pages 19 to 24	1-6

国際調査報告

国際出願番号　ＰＣＴ／ＪＰ０３／１２８４０

A.　発明の属する分野の分類（国際特許分類（ＩＰＣ））

Ｉｎｔ．Ｃｌ．⁷　Ｇ０６Ｆ１７／５０

B.　調査を行った分野

調査を行った最小限資料（国際特許分類（ＩＰＣ））

Ｉｎｔ．Ｃｌ．⁷　Ｇ０６Ｆ１７／５０

最小限資料以外の資料で調査を行った分野に含まれるもの

国際調査で使用した電子データベース（データベースの名称、調査に使用した用語）

C.　関連すると認められる文献

引用文献のカテゴリー＊	引用文献名　及び一部の箇所が関連するときは、その関連する箇所の表示	関連する請求の範囲の番号
Y	家長真行、外３名“制御処理ハードウェアの高位合成のための高速な面積／時間最適化アルゴリズム”、ＤＡシンポジウム２０００、情報処理学会、2000.07.17、Vol.2000、No.8、p.27-32	1-4,6
Y	Nakata, A. et al. "Deriving Parameter Conditions for Periodic Timed Automata Satisfying Real-Time Temporal Logic Formulas"、 Proc. of IFIP TC6/WG6.1 Int. Conf. on Formal Techniques for Networked and Distributed Systems(FORTE2001)、Kluwer Academic Publishers、2001.08、p.151-166	1-4,6

☒ Ｃ欄の続きにも文献が列挙されている。　☐ パテントファミリーに関する別紙を参照。

＊　引用文献のカテゴリー
「Ａ」特に関連のある文献ではなく、一般的技術水準を示すもの
「Ｅ」国際出願日前の出願または特許であるが、国際出願日以後に公表されたもの
「Ｌ」優先権主張に疑義を提起する文献又は他の文献の発行日若しくは他の特別な理由を確立するために引用する文献（理由を付す）
「Ｏ」口頭による開示、使用、展示等に言及する文献
「Ｐ」国際出願日前で、かつ優先権の主張の基礎となる出願の日の後に公表された文献
「Ｔ」国際出願日又は優先日後に公表された文献であって出願と矛盾するものではなく、発明の原理又は理論の理解のために引用するもの
「Ｘ」特に関連のある文献であって、当該文献のみで発明の新規性又は進歩性がないと考えられるもの
「Ｙ」特に関連のある文献であって、当該文献と他の１以上の文献との、当業者にとって自明である組合せによって進歩性がないと考えられるもの
「＆」同一パテントファミリー文献

国際調査を完了した日	国際調査報告の発送日
０６．１１．０３	25.11.03

国際調査機関の名称及びあて先	特許庁審査官（権限のある職員）		
日本国特許庁（ＩＳＡ／ＪＰ） 郵便番号１００－８９１５ 東京都千代田区霞が関三丁目４番３号	早川　学	５Ｈ	９６５２
	電話番号　０３－３５８１－１１０１　内線　３５３１		

様式ＰＣＴ／ＩＳＡ／２１０（第２ページ）（１９９８年７月）

国際調査報告

国際出願番号　PCT／JP05／12840

C（続き）．　関連すると認められる文献

引用文献のカテゴリー＊	引用文献名　及び一部の箇所が関連するときは、その関連する箇所の表示	関連する請求の範囲の番号
Y	GB 2317245 A(SHARP K.K.)1998.03.18、全文 & JP 10-116302 A	1-4,6
Y	若林一敏、外７名、“伝送用ＬＳＩを動作合成で開発，機能設計の期間が１／１０に短縮”、日経エレクトロニクス、日経ＢＰ社、1996.02.12、No.655、p.147-169	1-4,6
A	桶渡仁、外１名、“マルチプロセッサによる分散処理を意識した専用プロセッサ設計支援システムＳＹＡＲＤＳの構築”、情報処理学会研究報告、情報処理学会、1995.01.20、Vol.95、No.6(DA-73)、p.105-112	5
ＰＸ	北口智、外３名、“実時間制約を有する単一バスシステムのＪａｖａによるモデル化およびパラメトリックモデルチェッキングを用いた設計手法の提案”、情報処理学会研究報告、情報処理学会、2002.11.28、Vol.2002、No.113(SLDM-107)、p.19-24	1-6